Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Uživatelské rozhraní pro mikropočítač

Text User Interface for Microcontroller

2015 **Peter Gregorovič**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Peter Gregorovič**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Uživatelské rozhraní pro mikropočítač
Text User Interface for Microcontroler

Zásady pro vypracování:

Navrhněte jednoduché textové uživatelské rozhraní pro mikropočítače, připojené přes sériové rozhraní k PC tak, aby bylo ovládání možno provádět přes program hyperterminal nebo minicom.

1. Najděte a porovnejte dostupná textová rozhraní, vhodná pro mikropočítače.
2. Seznamte se s možnostmi textových terminálů.
3. Proveďte návrh rozhraní, kterým bude možno rozšířit i již hotové aplikace.
4. Implementujte uživatelské rozhraní jako knihovnu.
5. Realizujte prototypovou aplikaci pomocí navrženého rozhraní.
6. Zhodnoťte náročnost navrženého rozhraní, jeho stabilitu a složitost integrace do dalších aplikací.

Seznam doporučené odborné literatury:

[1] Stanislav Pechal, Monolitické mikropočítače, BEN - Technická literatura, ISBN:80-86056-30-9
[2] Text terminal HOWTO, http://www.linuxdoc.org/HOWTO/Text-Terminal-HOWTO.html

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

PROHLÁŠENÍ STUDENTA

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Praze dne 23. dubna 2015

podpis studenta

Poděkování

Rád bych poděkoval inženýru Petru Olivkovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce a za zapůjčení technických prostředků k vývoji a odladění výsledného produktu. Dále bych rád poděkoval svým kolegům za podporu, trpělivost a ochotu naslouchat i tématům, o kterých toho moc neví a za pomoc při korekturách textů této závěrečné práce.

Abstrakt

Tato bakalářská práce vznikla na základě potřeby obecné knihovny pro textové uživatelské rozhraní, zajišťující komunikaci mezi zařízením, řízeným mikrokontrolérem, a textovým terminálem.

Práce se zabývá přehledem terminálů, aplikací využívajících textová uživatelská rozhraní a zobecněním požadavků na textová rozhraní kladená. Dále pak definováním požadavků na budoucí knihovnu a její vlastní realizací včetně rozhraní, které ji přizpůsobí pro oba hlavní výrobce mikrokontrolérů Atmel i Microchip. V závěru se práce zabývá návodem a příklady pro vlastní integraci realizované knihovny pro textové uživatelské rozhraní.

Klíčová slova

Textové uživatelské rozhraní; Mikrokontrolér; Atmel; Microchip; Knihovna; Jazyk C; Escape sekvence; Terminál;

Abstract

This bachelor thesis was based on the needs of the general library for text user interface ensuring communication between devices controlled with a microcontroller and text terminal.

The work deals with an overview of terminals, applications that use text user interface and generalization of requirements for the text interfaces. Furthermore, by defining requirements for future library and its realization, including the interfaces that adjusts for both major manufacturers of microcontrollers, Atmel and Microchip. At the end the work deals with instructions and examples for integration of realized library for text user interface.

Key words

Text user interface; Microcontroller; Atmel; Microchip; Library; C language; Escape sequence; Terminal;

Seznam použitých zkratek

Zkratka	Význam
MC	Mikrokontrolér, en (Microcontroller)
GUI	Grafické uživatelské rozhraní, z en. (Graphics User Interface)
TUI	Textové uživatelské rozhraní, z en. (Text User Interface)
ASCII	en. (American Standard Code for Information Interchange)
AVR	Zkratka označující rodiny a typy mikrokontroléru výrobce Atmel
PIC	Zkratka označující rodiny a typy mikrokontrolérů výrobce Microchip
BIOS	en. (Basic Input-Output System)

V textu práce jsou uváděny názvy: Atmel, který je ochrannou známkou společnosti Atmel Corporation; Microchip, který je obchodní známkou společnosti Microchip Technology Inc.

Obsah

Úvod

V oblasti mikroelektroniky se v dnešní době využívá mnoho technických zařízení řízených mikrokontroléry. Většina z nich plní svůj účel bez nutnosti složitější interakce s jejich uživateli, případně zcela samostatně. Pro některé z nich by však byla komunikace alespoň na úrovni textového uživatelského rozhraní velkou přidanou hodnotou. Vývojáři vytvářející aplikace pro tyto zařízení se často soustředí pouze na primární funkce. Mnozí z vývojářů by pravděpodobně přivítali knihovnu, kterou lze lehce implementovat a splní běžné požadavky komunikace a řízení aplikace.

Tato bakalářská práce vznikla na základě potřeby obecné knihovny pro textové uživatelské rozhraní, zajišťující komunikaci mezi zařízením, řízeným mikrokontrolérem, a textovým terminálem.

Knihovna má umožnit řízení aplikace v zařízení pomocí přehledného a intuitivního ovládání v textovém režimu. Řízení aplikace s využitím knihovny bude umožňovat zobrazení i nastavení hodnot vlastností a spouštění funkcí aplikace v zařízení.

Knihovna má být implementovatelná do stávajících případně nově vznikajících aplikací s co nejmenším dopadem na existující kód aplikace. Vzhledem k využití jazyka C má být knihovna přeložitelná pro různé druhy mikrokontrolérů od hlavních výrobců Microchip i Atmel. Implementace a následná integrace knihovny má být paměťově co nejúspornější, vzhledem k poměrně omezeným paměťovým možnostem mikrokontrolérů.

Práce se zabývá přehledem terminálů, aplikací využívajících textová uživatelská rozhraní a zobecněním požadavků na textová rozhraní kladená. V dalších kapitolách se práce soustředí na definování požadavků pro budoucí knihovnu a její vlastní realizaci včetně rozhraní, které ji přizpůsobí pro dva významné výrobce mikrokontrolérů. V závěru se práce zabývá návodem a příklady pro vlastní implementaci realizované knihovny pro textové uživatelské rozhraní.

1. Uživatelská rozhraní

Pojem uživatelské rozhraní zahrnuje jakoukoliv interakci člověka s okolním světem. Pro potřeby této práce se však budu zabývat pouze rozhraními mezi člověkem v roli uživatele a technickými zařízeními ovládanými počítačem.

1.1 Možnosti lidské interakce

Člověk má, jak je známo, pět smyslů. Díky nim dokáže vnímat obrazy, zvuky, vůně, chutě a hmatové vjemy. V běžné praxi se však nepotkáme s tím, že by člověk počítač nějak ochutnával. Využití lidského hmatu v interakci s počítačem se v praxi zatím omezuje jen na úzkou, přesto neopomenutou skupinu uživatelů převážně se zrakovým postižením k rozpoznání Braillova písma. Vůně se prozatím v praxi také vůbec nevyužívá. Pokud už počítač cítíme, pak je zřejmě pozdě a něco se v něm spálilo nebo k tomu není daleko.

Nejběžnějším lidským smyslem, který uživatel v interakci s počítačem využívá, je zrak. Vnímání obrazu vytvářeného počítačem se postupně od signalizace stavu, přes tisky, textové a jednoduché grafické terminály až po dnešní plně barevné grafické počítače a tiskárny stalo tím nejdůležitějším vjemem. Jednoduchá zvuková signalizace, melodie, hudba až po synteticky vytvářené mluvené slovo je druhým nejdůležitějším vjemem, který člověk ke své interakci s počítačem využívá. Doplnění zrakových vjemů o zvuky pak dávalo i do dnes dává uživateli maximum podnětů, které od počítače ke své komunikaci potřebuje.

Na druhou stranu je člověk schopen počítači nabídnout mnohem méně. Dlouhodobě se využívala jen jediná možnost a to převod lidského pohybu na některý z ovládacích prvků počítače. Stisk tlačítka počítače ať už jako samostatného ovládacího prvku nebo jako součást klávesnice bylo po dlouhou dobu dostatečné a vystačíme si s tím i dnes. Komunikaci směrem od člověka k počítači se poslední dobou daří velmi úspěšně rozvíjet i v oblasti rukou psaného textu nebo mluveného slova. Dnes se už nikdo nepozastaví nad hlasovými příkazy v chytrých telefonech nebo rozšiřováním slova chápajících aplikací i na běžné osobní počítače.

1.2 Možnosti interakce počítačů

Jak už bylo naznačeno, počítač od člověka běžně vnímá poměrně málo a přesto je to v drtivé většině aplikací dostatečné. Schopnosti počítačů vnímat okolní svět i člověka v roli uživatele jsou však mnohem větší. Díky tomu, že nám elektrotechnika umožňuje převést základní měřitelné veličiny na elektrické signály zpracovatelné počítačem, se počítač nemusí omezit pouze na pohyb nebo zvuk vytvářený člověkem. Počítače dnes podobně jako člověk dokáží zpracovat obraz a rozhodovat se podle toho, co vidí. Tuto vlastnost mnozí znají z herních konzolí (Kinect). Poměrně dobrých výsledků se daří dosahovat i při měření aktivity lidského mozku a převedením zachycených vzorců na řídící signály pro počítač. V jednoduché formě je to již komerčně dostupná technologie pro ovládání her i dětských hraček.

1.3 Přetrvávající uživatelská rozhraní

I přesto, že rozmanitost komunikace člověka a počítače se od počátečních velmi omezených způsobů čím dále více podobá komunikaci člověka s jiným člověkem, bude se tato práce zabývat pouze jedním z nejstarších a do dnes velmi důležitým a stále využívaným rozhraním, kterým je textové uživatelské rozhraní.

2 Textová uživatelská rozhraní

Provedeným průzkumem dostupných textových uživatelských rozhraní v různých aplikacích a to nejen těch pro mikrokontroléry, bylo možné specifikovat několik kategorií rozhraní. Výsledky průzkumu jsem následně využil pro sestavení definice požadavků pro realizaci knihovny TUI.

Uživatelské rozhraní využívající pouze zobrazovaný a zadávaný text, tedy textové uživatelské rozhraní (TUI – Text User Interface) je jedním z nejstarších a přesto přetrvávajícím způsobem komunikace člověka jako uživatele a aplikací nějakého technického zařízení.

Prostředky využívané ke komunikaci uživatele a technického zařízení lze rozdělit do jednotlivých vrstev a to na (kromě samotného uživatele):

- terminál respektive terminálovou aplikaci na straně klienta,
- aplikaci využívající TUI na straně technického zařízení,
- samotné technické zařízení.

2.1 Terminály a terminálové aplikace

2.1.1 Jednoduché terminály

Označení jednoduché terminály (Dumb terminals) získávalo postupem času stále více terminálů jen proto, že se objevovaly terminály modernější a chytřejší proto tuto velmi často uváděnou kategorii nebudu dále rozepisovat [1].

2.1.2 Terminály textové

Jako „textový terminál“ (Text Terminal) lze označit zařízení, které obousměrně komunikuje s počítačem. Přenášeny jsou bajty (např. ASCII), které reprezentují (většinou) tisknutelné znaky. Znaky psané na klávesnici terminálu jsou odesílány do počítače a většina znaků zasílaných z počítače je zobrazována na displeji terminálu. Komunikace používá znaky kódované podle nějaké kódové stránky (většinou je to prvních 128 bajtů ASCII tabulky). K propojení terminálů a počítačů bylo dlouho používané propojení přes sériové porty (RS 232) kabelem. V dnešní době již počítače nejsou standardně vybaveny sériovým portem, je však možné využít různých převodníků USB-RS232, jež sériový port emulují. Vzhledem ke způsobu komunikace byly textové terminály často označované jako sériové terminály. Přesto terminál může k propojení na počítač využít i jiné spojení (modem, ethernet, apod.) [1] [2] [3].

Zřejmě nejrozšířenější řadou terminálů, jež se staly prakticky standardem i pro další výrobce, jsou terminály od společnosti DEC (Digital Equipment Corporation nyní Compaq) označované jako VT (např.: VT100, VT220, …) [4] [5].

Grafické možnosti textových terminálů jsou poměrně omezené a jsou závislé na použité znakové sadě. Jednoduché obrazce lze sestavit využitím standardních znaků jako šipky (<-, ->) nebo základní rámování. Některé textové terminály jsou schopny využít rozšířené znakové sady obsahující základní grafické tvary, s nimiž lze mnohem efektivněji zobrazit grafickou informaci.

2.1.3 Terminály v podobě tenkého klienta

Terminály označované jako tenký klient jsou plně grafické, přenášejí k uživateli veškerý komfort tak, aby měl pocit, že sedí u normálního počítače. Základním principem těchto terminálů zůstává skutečnost, že veškerý výpočetní výkon a práce se soubory je vykonávána na

počítači, ke kterému je terminál připojen - tím je většinou výkonný server. Tyto terminály mohou být tvořeny jednoduchým samostatným hardwarem s připojeným PC monitorem, ale stejně tak mohou být emulovány na jakkoliv výkonném PC pomocí k tomu určených terminálových aplikací. Tencí klienti jsou prakticky výhradně připojeni k serveru přes síťové rozhraní. Nejznámější protokoly pro přenos grafického uživatelského rozhraní jsou ICA protocol (Independent Computing Architecture) a RDP (Remote Desktop Protocol). Tencí klienti v podobě samostatného hardware jsou známí také pod označením síťový počítač nebo bezdiskový počítač [1].

Vzhledem k tomu, že tenký klient v terminálové terminologii nevyužívá textové, ale grafické uživatelské rozhraní, nebudu se mu v této práci dále věnovat.

2.1.4 Textové terminálové aplikace

Aplikace, které plní funkci textového terminálu, lze obecně označit jako emulátory textového terminálu. Přes různou funkcionalitu, kterou tyto aplikace nabízejí navíc, všechny shodně umí emulovat řadu původních terminálů a využít ke komunikaci základní druhy připojení a protokolů [1]. Mezi nejznámější aplikace v základních operačních systémech patří (řazeny abecedně):

Linux a X Windows	aterm, eterm, gnome, quake, konsole, Kuake, mrxvt, rxvt, rxvt-unicode, xfce4-terminal, Terminator, Terminology, tilda, wterm, xterm, Yakuake
MS Windows	AbsoluteTelnet, ConEmu, HyperTerminal, Indigo, mintty,-CygWin, PuTTY, RUMBA, SecureCrt, Tera Term, Terminator, TN3270+, TtyEmulator, Win32 console, ZOC terminal
Mac OS X	iTerm, MacWise, SecureCRT, SyncTERM, Terminal, Terminator, xterm, ZOC, ZTerm

Ve všech aplikacích emulujících terminály lze najít společné prvky ve způsobech připojení a emulovaných terminálech. Prakticky všechny emulátory umí komunikovat po sériovém portu a emulují nejrozšířenější terminály od výrobce DEC (VT52, VT100, VT102, VT220, …). Tyto společné vlastnosti inspirovaly i můj návrh knihovny textového rozhraní.

2.1.5 Proč dále využívat textová rozhraní

Textová rozhraní se postupně vytratila ze všech zařízení a systémů určených pro širokou laickou veřejnost a byla postupně nahrazena grafickými aplikacemi nebo webovými prohlížeči. Zastoupení textových rozhraní bylo zachováno hlavně u zařízení obsluhovaných odborníky v dané oblasti toho kterého zařízení. Textová rozhraní mají výrazně nižší nároky na programovou i datovou paměť zařízení než grafická rozhraní. Uživatel může podle druhu aplikace rychle a přehledně spravovat vlastnosti i ovládat konkrétní aplikaci. Textové rozhraní je většinou pro tyto činnosti velmi dobře optimalizováno. Textové rozhraní nemůže svými možnostmi konkurovat plně grafickým rozhraním (určitě v něm nelze namalovat podrobný graf), přesto si své pole působnosti stále udržuje. Textové rozhraní je nezastupitelné nejen u mnoha technických zařízení řízených mikrokontrolérem, ale i u složitějších zařízení řízených procesory a také u velkých operačních systémů. Administrátoři by si ani nedokázali představit, že by neměli k dispozici řádkový příkazový režim u serverových i klientských operačních systémů (Linux, MS Windows, UNIX apod.) [1] [6].

2.2 Současné aplikace využívající TUI

Aplikace využívající textové uživatelské rozhraní jsem zařadil do kategorií podle jejich charakteristiky chování:

- Aplikace konzolové s menu
- Aplikace s příkazovým řádkem
- Aplikace celoobrazovkové

2.2.1 Aplikace konzolové s menu

Konzolové aplikace se vyznačují jednoduchým strukturovaným menu a výpisem veškerých výstupů aplikace na „nekonečný“ displej a to včetně voleb menu. Informace jsou postupně rolovány a uchovává se jen poslední část podle nastavení. Pomocí voleb je možné zobrazovat a volat jakékoliv vnitřní procesy, které zajistí vykonání požadované funkcionality a zobrazení jejího výsledku, čímž lze zobrazit jakoukoliv informaci. Volbou lze také vyvolat zadání nové hodnoty vlastnosti. Jednotlivé funkce aplikace se volí ve strukturovaném menu, nikoliv přímo zápisem. Pokud jsou strukturované volby (menu aplikace) připraveny kvalitně, může být aplikace velice intuitivní a nebude od uživatele vyžadovat její důkladnou znalost.

2.2.2 Aplikace s příkazovým řádkem

Aplikace s příkazovým řádkem se vyznačují zápisem příkazů včetně argumentů do příkazové řádky. Rozpoznání a vykonání příkazů pak realizuje veškerou funkcionalitu dané aplikace. Výstupy aplikace jsou realizovány podobně jako u konzolové aplikace na „nekonečný“ displej. Aplikace s příkazovým řádkem mohou být poměrně rozsáhlé (např. MS PowerShell), ale mohou být také velmi triviální, kdy mají pouze jedinou funkci a argumenty jsou dodány již při spuštění aplikace.

2.2.3 Aplikace celoobrazovkové

Aplikace celoobrazovkové využívají k zobrazení informací a práci s nimi pevně danou plochu displeje klientské aplikace nebo terminálu. Na rozdíl od předchozích dvou tedy při běžném provozu donekonečna nerolují zobrazované informace. Asi nejrozšířenějším příkladem pro takovou aplikaci je BIOS osobních počítačů. V těchto aplikacích jsou informace většinou strukturovány do stránek a položek na stránce. Je tedy možné zobrazení více hodnot najednou na jedné stránce. Zadávání hodnot a vykonání funkcionality se provádí najetím a volbou dané položky na stránce, která reprezentuje hodnotu vlastnosti respektive funkci aplikace [6].

2.2.4 Srovnání aplikací využívajících TUI

Všechny popsané kategorie nějakým způsobem zobrazují texty a do textu převedené hodnoty vlastností. V následné tabulce je uveden přehled základních vlastností pro popsané tři kategorie aplikací.

Vlastnost \ Aplikace	konzolové s menu	s příkazovým řádkem	Celoobrazovkové
Rolování výstupu	ano	ano	ne
Využití plochy displeje	ne*	ne*	ano
Spouštění funkcí napsáním příkazu	ne	ano	ne
Strukturované menu	ano	ne	ano
Zobrazení více hodnot najednou	ano*	ano*	ano
Automatická aktualizace zobrazení	ne	ne	ano

* pouze s využitím rolování vypsaného textu

Všechny druhy aplikací využívající textové rozhraní mají v obecné rovině společné tyto základní funkce (případně jejich kombinace):

- Zobrazení textu – popisů, statických informací apod.
- Zobrazení textové reprezentace hodnoty vlastnosti různých datových typů
- Zadání textové reprezentace hodnoty vlastnosti různých datových typů
- Spouštění funkcí aplikace

2.3 Oblast technických zařízení

Oblastí, na kterou se tato bakalářská práce soustředí, jsou technická zařízení ovládaná nebo alespoň komunikující pomocí mikrokontrolérů. Jednočipové počítače označované jako mikrokontroléry v sobě obsahují vše, co má počítač obsahovat podle harvardské definice architektury počítače. Mimo to také obsahují vše, co je potřebné, aby mohly svou funkci vykonávat po připojení ke zdroji energie a to i bez dalších elektronických součástek. Příkladem pro to, jak nenápadný a malý může takový počítač být, je čipová platební karta. Typů mikrokontrolérů je velké množství. Dělit je lze nejen podle výrobce, ale hlavně podle jejich vlastností a schopností. Kritéria jako velikost pamětí (jak programové tak i datové), taktovací frekvence, šíře sběrnice a instrukční sada vypovídají jen o procesoru čipu. Důležitější kritéria pro výběr mikrokontrolérů a nasazení do konkrétního technického zařízení jsou periférie, které čip obsahuje, nebo díky kterým se na další dokáže připojit [2].

Tato bakalářská práce se tedy omezí pouze na technická zařízení, které v sobě obsahují mikrokontrolér schopný komunikovat s terminálem respektive terminálovou aplikací na straně klienta. Komunikací v tomto smyslu se rozumí nejen propojení technického zařízení a terminálu, ale i vzájemné porozumění komunikačnímu protokolu při obousměrném přenosu textové informace.

V současnosti je realizováno nespočet zařízení různé úrovně složitosti a kvality, které využívají textové uživatelské rozhraní. V oblastech profesionálních zařízení (např. diagnostika a nastavení řídících jednotek motorů) není zmapování TUI snadno dostupné ani jednoduché. Nejdostupnější jsou zařízení v oblasti informačních technologií, a to od jednoduchých měřících a řídících zařízení až po složité systémy řízení sítí (např. síťové prvky Cisco).

Většina technických zařízení v oblasti zájmu bakalářské práce využívá TUI k diagnostice a nastavení vlastností daného zařízení, případně k manuálnímu ovládání funkcí zařízení, které mohou být při běžném provozu zcela automatické. Mohou se však vyskytovat i zařízení, u kterých veškerá funkcionalita zcela spoléhá na textové rozhraní. Až na výjimky je většina technických zařízení vybavena alespoň základními prvky ovládání (např. tlačítka Zapnout/Vypnout; Start/Stop; apod.), stejně tak alespoň jednoduchou zpětnou vazbou o stavu zařízení (od LED signalizující stav až po LCD displej s údaji). Jako příklad by se dala uvést domácí meteostanice, která sice pro uživatele zobrazuje vše, co kdy uživatel bude potřebovat, ale pro její kalibraci je k dispozici servisní mód, využívající textové uživatelské rozhraní.

Zařízení spadající do zájmové oblasti nezahrnují pouze nově vytvářené aplikace do zařízení, která budou využívat TUI, ale i stávající zařízení, která lze o tuto komunikační schopnost rozšířit a to v případech, kdy to pro ně bude vytvářet (mít) přidanou hodnotu.

3 Návrh požadavků na textové rozhraní

Z provedeného průzkumu a následně zobecněných požadavků v souladu se zadáním lze nyní specifikovat konkrétní požadavky na knihovnu funkcí zajišťující textové uživatelské rozhraní. Pro lepší přehlednost jsou požadavky rozděleny do kategorií popsaných v samostatných podkapitolách.

3.1 Technologické požadavky

V souladu se zobecněnými požadavky je nutné zajistit, aby zdrojové kódy napsané v jazyce C bylo možné přeložit pro dva významné výrobce mikrokontrolérů (Atmel i Microchip). Pro 8 i 32 bitové mikrokontroléry výrobce Atmel budou zdrojové kódy knihovny přeložitelné v jejich vývojovém prostředí Atmel Studio (verze 6 a vyšší). Pro mikrokontroléry od výrobce Microchip budou zdrojové kódy přeložitelné v jejich vývojovém prostředí MPLAB X IDE (verze 2.20 a vyšší), kompilátor XC8 (verze 1.32 a vyšší) respektive kompilátory XC16 a XC32 podle třídy mikrokontroléru [7] [8] [9].

Textová komunikace knihovny nebude závislá na způsobu připojení terminálu ani na obousměrném komunikačním protokolu zajišťujícím přenos textové informace. Knihovna bude přijímat příkazy i text z terminálu a stejně tak bude posílat do terminálu text i příkazy ve formátu kompatibilním s terminálem VT-100 s využitím jen nejnutnějších řídících sekvencí [4].

3.2 Požadavky na vzhled

Knihovna, při implementaci do aplikací dalších vývojářů, zajistí možnost definovat vlastnosti celého menu, jednotlivých stránek a vlastnosti položek na stránkách.

Zobrazování menu, jednotlivých stránek a jejich položek bude realizováno jako u aplikací celoobrazovkových. Bude tedy využita celá plocha displeje, který bude rozdělen tak, aby byly přehledně zobrazeny všechny stránky, položky aktivní stránky i jejich hodnoty a to včetně případných popisů položek (nápovědy). Na displeji bude vyhrazeno místo pro titulní popis aplikace, pro zadávání nových hodnot a také místo pro zprávy z aplikace.

Knihovna zajistí zobrazení textové reprezentace jakéhokoliv datového typu, přičemž překlad hodnot do textové reprezentace pro účely zobrazení i případné změny zajistí funkce v aplikaci implementující knihovnu [6].

3.3 Požadavky na funkcionalitu

Pohyb mezi stránkami bude realizován stiskem kurzorových tlačítek (vlevo/vpravo). Pohyb mezi jednotlivými položkami bude realizovaný stiskem kurzorových tlačítek (nahoru/dolů). Změna aktuálně vybrané položky nebo spuštění funkce bude provedeno stiskem tlačítka Enter.

Pro zadávání hodnot vlastností položek bude zpracován řádkový editor umožňující úpravu textové reprezentace hodnoty jakéhokoliv datového typu. Vlastní překlad textové reprezentace na správný datový typ zajistí funkce položek realizované v aplikaci implementující knihovnu.

Knihovna umožní automatickou aktualizaci zobrazení hodnot položek na stránce v nastaveném časovém intervalu [6].

4 Návrh a realizace knihovny TUI

Realizace knihovny pro tuto bakalářskou práci stejně jako realizace jakéhokoliv jiného softwarového díla prošla několika standardními fázemi vývoje. Seznámení se s problematikou v dané doméně, analýzou, specifikací přesného zadání a vlastní realizací. Následně také popisem realizace, možnostmi implementace a návodu použití.

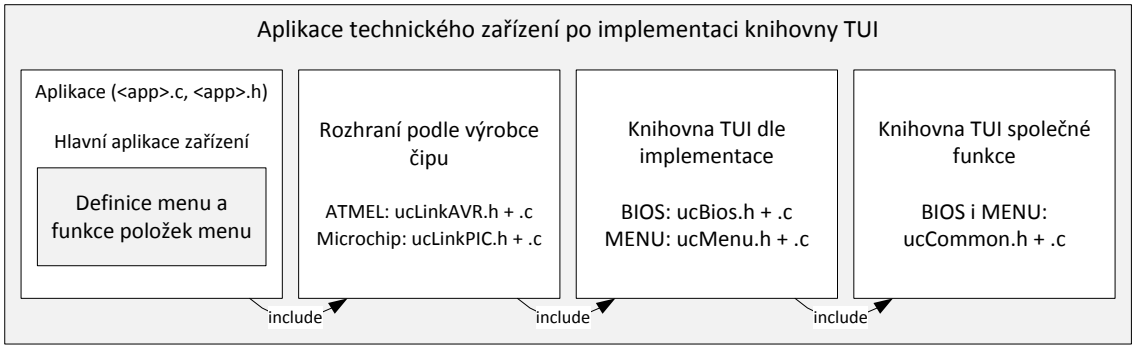
Při realizaci knihovny jsem v počátku přistoupil k návrhu předpokládaného vzhledu a zároveň s tím jsem vytvářel i struktury potřebné k definování všech vlastností a funkcí menu, nutných k výslednému zobrazení i interakci s uživatelem.

Následně bylo nutné rozhodnout, jakým způsobem bude možné knihovnu textového uživatelského rozhraní implementovat do stávající či nově vznikající aplikace. V průběhu realizace vykrystalizovaly dva možné způsoby implementace. Jedním z nich (*ucBios*) je využitím velmi podobný jako BIOS u PC. Při tomto způsobu se proces knihovny spouští na základě nějaké události (např. tlačítko, spínač) a probíhá, dokud není uživatelem ukončen. Druhým způsobem implementace (*ucMenu*) je proces knihovny opakovaně volán z hlavního vlákna aplikace. Umožňuje interakci s uživatelem po celou dobu a probíhá paralelně s hlavní aplikací technického zařízení [6].

Pro snadnější a rychlejší implementaci knihovny TUI jsem pro její podporu vytvořil dvě rozhraní (sady nízko úrovňových funkcí pro řízení HW) orientovaná na významné výrobce (Atmel i Microchip). Tato rozhraní v sobě obsahují vzorovou implementaci všech povinných funkcí, které knihovna TUI vyžaduje.

Výsledná aplikace tak pro využití knihovny připojí mnou vytvořený soubor pro rozhraní podle výrobce čipu (či klon s vlastní implementací rozhraní). V daném rozhraní se pak připojí jeden ze dvou souborů knihovny podle požadovaného způsobu chování a implementace. V souboru hlavní aplikace, případně v samostatném souboru, je pak ještě potřebné vytvořit definici menu a všechny případné funkce položek na stránkách menu. Blokově je způsob implementace znázorněn na obrázku 5.1.

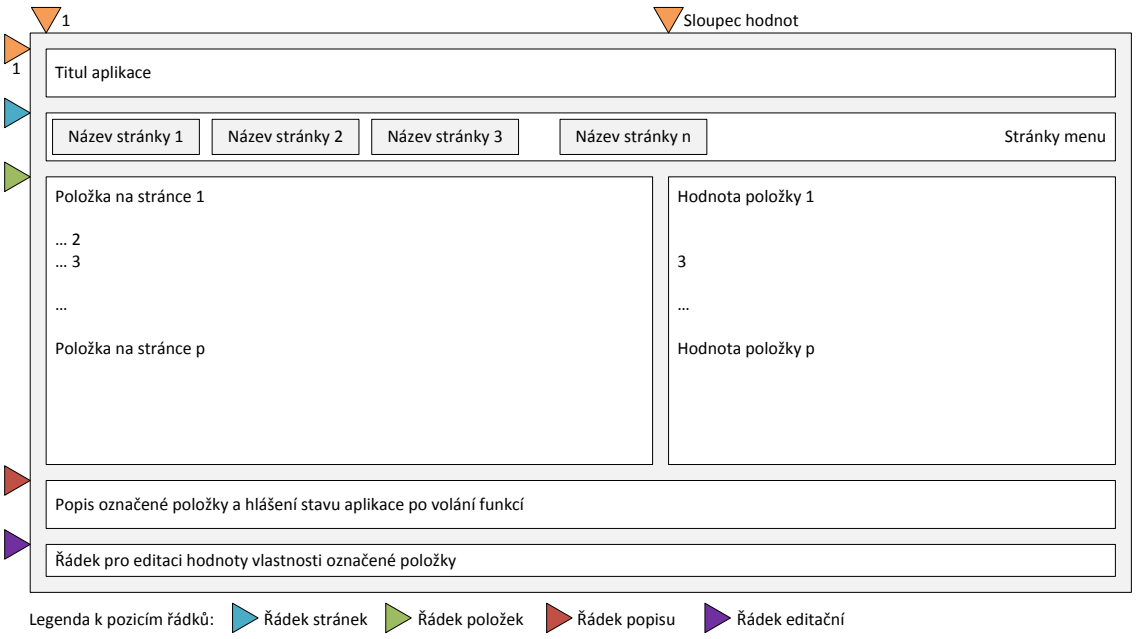
Podrobný popis implementace knihovny TUI do nové či stávající aplikace je popsán v samostatné kapitole 5, a to včetně řady příkladů pro všechny varianty rozhraní i knihovny.



Obrázek 4.1: *Integrace knihovny TUI*

4.1 Návrh a realizace výsledného vzhledu

Pro co nejpřehlednější zobrazení stránek a položek v nich jsem se rozhodl pro uspořádání údajů na ploše displeje znázorněném na obrázku číslo 4.2.



Obrázek 4.2: *Rozložení údajů na ploše*

Vzhledem k tomu, že textové terminály zobrazují neproporcionální znaky, jsou všechny znaky uspořádány v pevně definované mřížce o rozměru podle typu terminálu či terminálové aplikace (standardně 80 × 25 znaků). Tato vlastnost přináší řadu výhod při uspořádání zobrazení a odpadají starosti o přesné formátování. Nevýhodou proti proporcionálnímu písmu je pak schopnost zobrazit méně textu na danou plochu.

Jak je z obrázku patrné, plocha displeje pro zobrazení menu aplikace je rozdělena na samostatné bloky. Všechny texty v blocích i pozice bloků budou zadány v definici menu v aplikaci implementující knihovnu TUI. V bloku titulu aplikace se zobrazí text identifikující a popisující aplikaci. Blok titulu může být víceřádkový a poskytuje tak dostatek prostoru neomezený maximálním počtem znaků na řádku. V bloku stránek menu se pak odděleně zobrazují názvy jednotlivých stránek menu, přičemž název aktuální (vybrané) stránky se zobrazí jako invertovaný a neaktivní jako podtržený.

Blok s položkami na stránce menu obsahuje sloupec pod sebou zobrazených názvů položek, kdy aktuální (vybraná) položka se zobrazuje invertovaná. Položky mohou být i pouze textové (popisné) a tedy i prázdné. V případě prázdné položky se zobrazí prázdný řádek a takovou položku nelze označit. Blok hodnot jednotlivých položek začíná na pozici zadané v definici menu a obsahuje sloupec hodnot jednotlivých položek, pokud tyto položky hodnotu obsahují. V případech kdy je položka pouze textová (i prázdná) nebo jen pro spuštění funkce, pak se na řádku takové položky žádná hodnota nezobrazí.

V bloku pro popis položek se zobrazí popisná informace (nápověda) k aktuální položce na stránce menu a to v případech, kdy takový popis položka obsahuje, jinak bude řádek

prázdný. Ve stejném bloku se zobrazuje i informace vrácená zápisovou funkcí položky menu, po tom co je hodnota položky změněna nebo byla volbou spuštěna funkce. Mohou to být informace o úspěšném provedení změny hodnoty, o chybě při zadání nebo hlášení výsledků provedených operací po spuštění funkce. Blok může být víceřádkový a poskytuje tak dostatek prostoru neomezený pouze maximálním počtem znaků na řádku.

Posledním blokem je editační řádek, který je určen pro zadávání nových hodnot položek menu. Blok bude využit u položek menu umožňující změnu hodnoty. Při editaci hodnoty se na začátku řádku zobrazí text reprezentující editační mód (takzvaný prompt) a za ním pak původní hodnota a následně editovaná hodnota. Editaci hodnoty lze přerušit stiskem klávesy Escape nebo novou hodnotu potvrdit klávesou Enter.

4.2 Realizace definice menu

Pro možnost uložení všech informací o menu bylo nutné definici menu rozdělit do jednotlivých na sebe navázaných částí. V objektovém jazyce bychom mluvili o jednotlivých objektech, ale v jazyce C zůstaneme u definování typů a struktur [7]. Části v definici menu jsou:

- Definice menu - obsahuje všechny vlastnosti platné obecně pro celé menu, ukazatele na základní povinné funkce a seznam (pole) stránek menu.
- Stránka menu - obsahuje všechny vlastnosti platné pro celou stránku menu a seznam (pole) položek na stránce menu, případně ukazatel na funkci stránky.
- Položka na stránce menu - obsahuje vlastnosti položky a případné ukazatele na čtecí a zápisovou funkci, jejichž existence definuje typ položky.

Celá definice je statická a proto by měla být uložena v programové části paměti a nezabírat prostor v datové paměti, která je u mikrokontrolérů velmi malá [7].

Všechny tři části v definici menu obsahují definované ukazatele na funkce pro čtení a zápis hodnoty. Oba typy funkcí vrací stejnou hodnotu a tou je struktura *ucResponse* viz 4.1.

```
typedef struct ucResponse                                    (4.1)
{
    enum ucResponseType ResponseType;
    char *ResponseText;
} ucResponse;
```

Díky této struktuře je možné číst nejen text, ale také stav po vykonání dané funkce. Stav *BiosExit* z výčtu *ucResponseType* využívá jen jedna z implementací knihovny (*ucBios*) a to k ukončení práce knihovny a návratu do hlavního vlákna aplikace. Ostatní stavy ovlivňují, jakým způsobem se zobrazí návratová zpráva. Výčet stavů v návratové hodnotě (soubor *ucCommon.h*) viz 4.2.

```
enum ucResponseType                                          (4.2)
{
    None = 0,           //show text as normal text
    Information = 1,    //show text as underscored
    Warning = 2,        //show text as bold
    Error = 3,          //show text as bold, blinking
    BiosExit = 4,       //exit menu
};
```

Čtecí funkce vrací ve struktuře kromě typu i textovou reprezentaci hodnoty položky na stránce menu. Zápisová funkce respektive volání funkce aplikace pak v textové části může vracet text se zprávou o průběhu a vykonání funkce, případně chybové hlášení.

Definice ukazatelů na čtecí a zápisové funkce (soubor *ucCommon.h*) viz 4.3.

```
typedef ucResponse(*ucItemRead)(void);
typedef ucResponse(*ucItemWrite)(char* text);
```
(4.3)

4.2.1 Definice menu

Menu jako celek je definováno ve statické struktuře obsahující:

- Text titulku menu a promptu *(MenuTitle, EnterPrompt)*
- Ukazatele na funkce pro inicializaci a destrukci menu *(Initialization, Dispose)*
- Pozice řádků a sloupce *(PagesLine, ItemsLine, DescrLine, EnterLine, ValuesCol)*
- Ukazatele na povinné funkce rozhraní *(ReceiveCount, ReceiveChar, TransmitChar, MenuIntToDecStr, RefreshCounterRead, RefreshCounterReset, MenuWaitOrSleep)*
- Pole se seznamem stránek menu *(MenuPages)*

Knihovna počítá s tím, že pole stránek obsahuje alespoň jednu stránku, neboť by jinak menu nemělo smysl, proto prázdný seznam stránek knihovna nijak nekontroluje. Při prázdném seznamu by tak mohlo dojít k neočekávanému chování knihovny. Všechny ukazatele na funkce využívají definované typy ukazatelů a jsou tak už při překladu kódu typově kontrolovány.

Struktura definice menu (soubor *ucCommon.h*) v kódu 4.4.

```
typedef struct ucMenu
{
    const char *MenuTitle;          //Muze byt viceradkovy -> posunout radky vseho
    const ucMenuPage *MenuPages;    //Seznam stranek menu
    ucItemRead Initialization;      //Funkce pro inicializaci menu
    ucItemRead Dispose;             //Funkce pro destrukci menu
    unsigned char PagesLine;        //Radek, na kterem jsou zobrazeny tituly stranek (zalozky)
    unsigned char ItemsLine;        //Radek, na kterem zacina prvni polozka stranky menu
    unsigned char DescrLine;        //Radek, na kterem zacina popis polozky menu
    unsigned char EnterLine;        //Radek, na kterem bezi editor hodnot a zobrazeni vysledku
    unsigned char ValuesCol;        //Sloupec, od ktereho jsou zobrazeny hodnoty polozky menu
    char *EnterPrompt;              //Text promptu zobrazenym pred editovanou hodnotou
    ucReceiveCount ReceiveCount;    //Funkce pro poctu znaku na vstupu rozhrani
    ucReceiveChar ReceiveChar;      //Funkce pro nacteni jednoho znaku ze vstupu
    ucTransmitChar TransmitChar;    //Funkce pro odeslani jednoho znaku na vystup
    ucIntToDecStr MenuIntToDecStr;  //Funkce pro preklad int na text
    ucRefreshCounterRead RefreshCounterRead;    //Funkce pro nacteni hodnoty citace
    ucRefreshCounterReset RefreshCounterReset;  //Funkce pro reset hodnoty citace
    ucWaitOrSleep MenuWaitOrSleep;  //Funkce pro cekani na urceny cas
} ucMenu;
```
(4.4)

4.2.2 Definice stránky menu

Struktura definující stránku menu je poměrně jednoduchá a obsahuje pouze vlastnosti:

- Název stránky a interval pro aktualizaci jejích hodnot
- Ukazatel na funkci stránky
- Pole se seznamem položek na stránce menu

Každá stránka umožňuje aktualizaci hodnot jednotlivých položek a to v uvedeném intervalu (*RefreshTime*). V případě, že je tato hodnota nastavena na nulu, nebude na stránce docházet k aktualizaci hodnot položek nikdy. Stránka rovněž obsahuje vlastní ukazatel na zápisovou funkci. Funkci stránky lze spustit klávesou Enter, pokud je ukazatel aktuální položky přesunut do řádku se seznamem stránek a jako aktivní je označena stránka s definovanou funkcí.

Struktura definice stránky menu (soubor *ucCommon.h*) viz kód 4.5.

```
typedef struct ucMenuPage                                                    (4.5)
{
    char *PageTitle;                //Nazev (titulek) stranky
    int RefreshTime;                //Hodnota citace pri ktere se ma aktualizovat. 0=nikdy
    const ucMenuItem *MenuItems;    //Seznam polozek na strance menu
    ucItemWrite Write;              //Funkce stranky
} ucMenuPage;
```

4.2.3 Definice položky na stránce menu

Struktura popisující položku na stránce je poměrně jednoduchá (viz kód 4.6, ze souboru *ucCommon.h*) a obsahuje pouze:

- Název a popis položky na stránce menu
- Dva ukazatele na funkce (čtecí a zápisovou)

Právě přítomnost ukazatelů vytváří základní čtyři druhy položky na stránce, jimiž jsou:

- Text – text nebo také prázdný řádek popisující respektive oddělující bloky položek
- Pouze čtená hodnota – položka s hodnotou, kterou nelze měnit (Read)
- Hodnota položky – položka s hodnotou, kterou lze číst i měnit (Read + Write)
- Funkce – položka, reprezentující funkci menu (Write)

```
typedef struct ucMenuItem                                                    (4.6)
{
    char *ItemName;          //Nazev polozky na strace menu
    char *ItemDescription;   //Popis polozky na strance menu
    ucItemRead Read;         //Funkce pro cteni textove reprezentace hodnoty
    ucItemWrite Write;       //Funkce pro zapis hodnoty nebo volani funkce
} ucMenuItem;
```

4.3 Realizace výkonné části

Jak už bylo v úvodu této kapitoly popsáno, je výkonná část knihovny realizovaná pro dvě varianty implementace (BIOS a MENU). Funkcionalita i výsledný vzhled poskytovaný knihovnou je u obou variant identický. Ačkoliv jsou základní funkce pro řízení terminálu u obou variant knihovny stejné, zbytek funkcí se liší zejména podle požadovaného chování dané varianty.

Hlavičkové soubory obou verzí odkazují na stejné definice typů všech společných struktur (např. definice menu), funkcí a výčtů. Stejně pak odkazují i na definice ukazatelů na všechny povinné funkce rozhraní (soubor *ucCommon.h*) viz kód 4.7.

```
typedef int(*ucReceiveCount)(void);                                          (4.7)
typedef unsigned char(*ucReceiveChar)(void);
typedef void(*ucTransmitChar)(unsigned char chr);
typedef int(*ucRefreshCounterRead)(void);
typedef void(*ucRefreshCounterReset)(void);
typedef void(*ucWaitOrSleep)(int miliseconds);
typedef void(*ucIntToDecStr)(int value, char *text);
```

Implementace povinných funkcí vytváří rozhraní (sadu nízko úrovňových funkcí pro řízení HW) mezi knihovnou a konkrétní technickou realizací zařízení. Obsahuje ty nejzákladnější komunikační funkce pro přečtení a odeslání znaku, funkce pro čítač pro aktualizace stránky a funkci pro uspání vlákna na daný čas. Obsah všech těchto povinných

funkcí je závislý jak na konkrétním mikrokontroléru, tak na technickém řešení jeho zapojení a propojení na terminál. Vyjmutí povinných funkcí z realizace knihovny umožňuje této knihovně stát se daleko flexibilnější s možností jejího nasazení ve výrazně širším okruhu aplikací.

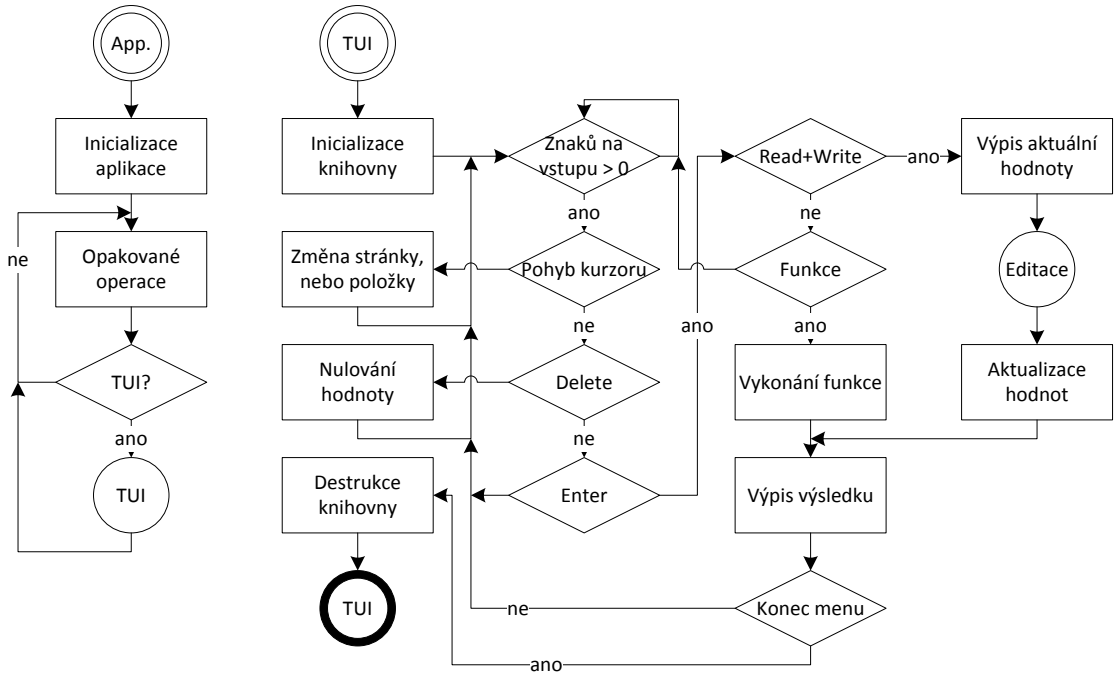
4.3.1 Stejné funkce u knihovny BIOS i MENU

Vnitřní privátní funkce pro ovládání terminálu jsou společné pro obě varianty implementace knihovny. Zajišťují odesílání textu do terminálu, smazání celého displeje, řádku nebo jen jeho části a nastavení atributů pro zobrazovaný text. Podrobný popis funkcí a jejich atributů je uveden v příloze A, včetně diagramů jejich vzájemného propojení. Tyto funkce jsou jediné, které do terminálu odesílají řídící (Escape) sekvence [4]. Seznam v knihovně nejen použitých, ale i všech řídících sekvencí terminálu VT100 je uveden v příloze C. Seznam definic ukazatelů řídících funkcí (soubory *uBios.h* nebo *uMenu.h*) viz 4.8.

```
void _SendText(const ucMenu *menu, const char* buffer);                    (4.8)
void _ClearDisplay(const ucMenu *menu);
void _SetPosition(const ucMenu *menu, unsigned char yLine, unsigned char xChar);
void _ClearLineFromCursor(const ucMenu *menu);
void _ClearLine(const ucMenu *menu, unsigned char yLine);
void _SetAttribute(const ucMenu *menu, char bold, char underscored, char blinking, char invert);
```

4.3.2 Knihovna realizovaná jako BIOS

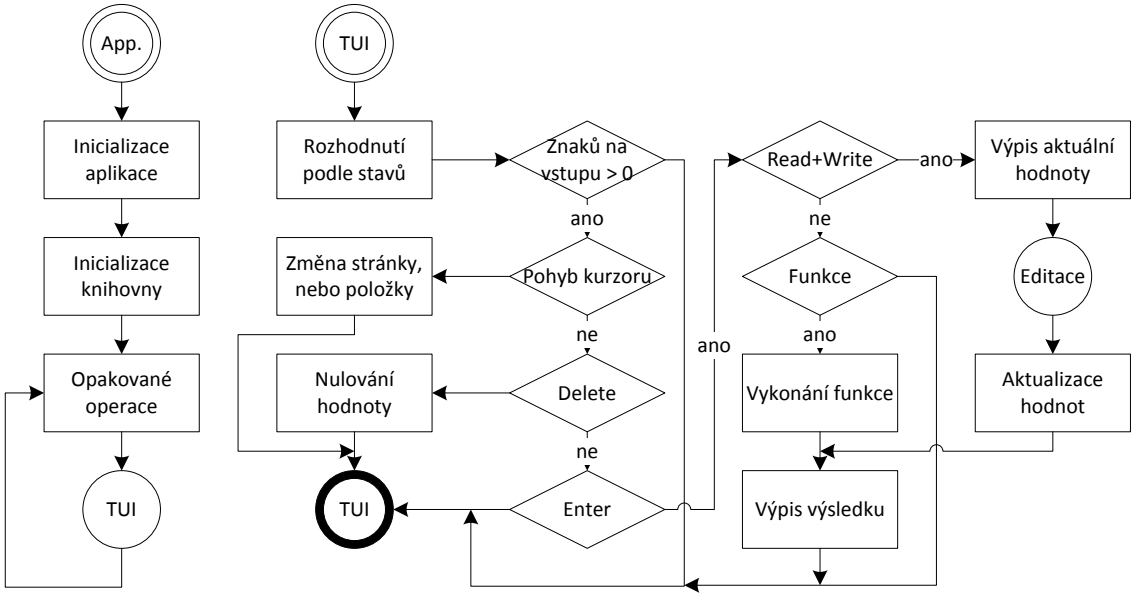
Varianta implementace knihovny realizovaná jako BIOS má jedinou veřejnou funkci, která je použita na spuštění knihovny (RunMenu). Tato funkce zajistí volání funkcí pro inicializaci při startu a uvolnění při ukončení svého běhu. Mezitím se stará o veškerou komunikaci s terminálem uživatele, výpis i aktualizaci informací na displeji terminálu a obsluhu dat, co z terminálu od uživatele přicházejí. Jakákoliv zápisová funkce může ve své návratové hodnotě obsahovat typ pro ukončení menu, který je pak signálem pro destrukci knihovny a návrat do hlavní aplikace. Jak je z tohoto popisu zjevné, menu plně přebírá řízení a hlavní vlákno aplikace po dobu běhu knihovny stojí. Jediná možnost pro zachování funkcionality řízení technického zařízení současně s jeho nastavováním je využití přerušení.



Obrázek 4.3: *Diagram implementace TUI jako BIOS*

4.3.3 Knihovna realizovaná jako MENU

Ve variantě implementace knihovny realizované jako MENU je nutné knihovnu inicializovat dříve podobně jako inicializaci hlavní aplikace. Ve smyčce hlavní aplikace je pak periodicky volána funkce knihovny *DoMenuActions*. Tato funkce pak podle stavu knihovny zajistí veškerou komunikaci s terminálem uživatele, výpis i aktualizaci informací na displeji a obsluhu všeho, co z terminálu na straně uživatele přichází. Tuto variantu implementace nelze ukončit, proto není nutné implementovat funkci pro destrukci menu. Protože po připojení terminálu nelze určit, v jakém stavu se běžící aplikace nachází, je v této variantě přidána funkcionalita na aktualizaci zobrazení menu. Po dvojitém stisku klávesy Escape se aktualizují informace celého displeje.



Obrázek 4.4: *Diagram implementace TUI jako MENU*

4.4 Realizace rozhraní podle výrobce mikrokontrolérů

Vzhledem k rozdílnosti mikrokontrolérů nejen z hlediska jejich technických parametrů, ale hlavně z hlediska výrobcem dodávaných vývojových prostředí a překladačů zdrojového kódu, jsem pro usnadnění a urychlení integrace vytvořit i dvě varianty vzorového rozhraní. Obě varianty jsou orientované na konkrétního výrobce čipů a zohledňují tak specifika základních vlastností mikrokontrolérů i standardního vývojového prostředí a překladačů daného výrobce. Ve vytvořených vzorových rozhraních je komunikace s terminálem prováděna přes rozhraní USART. Lze však při konkrétním nasazení využít jakéhokoliv dostupného komunikačního kanálu (např. I2C, Ethernet, Bluetooth, apod.). Podobně pak časování, určené pro aktualizaci stránek, není exaktně nastavené podle krystalu nebo přepočtem frekvencí, ale spíše odhadem na počet cyklů nekonečné smyčky hlavního vlákna aplikace, respektive počtem dotazů na počet příchozích znaků. Toto řešení není z časového hlediska naprosto přesné, ale k účelu automatické aktualizace v prototypové aplikaci naprosto dostačuje. Nastavení přesného časování je plně na vývojáři integrujícím knihovnu TUI, který může využít všech prostředků, které mu jím zvolený mikrokontrolér nabízí.

4.5 Popis možností a ovládání knihovny TUI

Výsledná aplikace s integrovanou knihovnou TUI se chová jako celoobrazovková aplikace a tomu také odpovídá i její ovládání. Knihovna zajišťuje zobrazení všech bloků informací na ploše displeje (viz obr.: 4.2). Ve variantě BIOS se celá plocha aktualizuje pouze po spuštění knihovny. Ve variantě MENU lze obnovení celé plochy vynutit dvojitým stiskem klávesy Escape.

V řádku stránek jsou za sebou zobrazeny názvy stránek menu. Stránky jsou odděleny znakem „|“. Název aktuální stránky je na řádku zobrazen inverzním písmem, ostatní stránky jsou podtržené. V případě, kdy není aktivní žádná z položek na stránce, ale aktivní je přímo stránka, název stránky se zobrazuje inverzně a tučně.

Aktivní položky na stránce jsou zobrazovány inverzním písmem, ostatní písmem normálním. Vedle položek, které mají uvedený ukazatel na čtecí funkci, se zobrazují jejich hodnoty. Atributy písma pro zobrazení hodnot odpovídají typu v návratové hodnotě čtecí funkce.

Atributy písma pro zobrazení hodnot a zpráv po vykonání funkcí jsou v knihovně nastaveny podle typu v návratové hodnotě následovně:

- None normální text
- Information podtržený text
- Warning tučný text
- Error blikající tučný text.

4.5.1 Ovládání na stránkách menu

Změnu aktivní stránky lze provést stiskem kurzorových kláves šipka vpravo respektive šipka vlevo. Pohyb vpravo je cyklický, takže z poslední stránky se po dalším stisku klávesy (šipka vpravo) přesune aktuální stránka na první. Pohyb vlevo cyklický není, končí na první stránce.

Pohyb po jednotlivých položkách na stránce se provádí stiskem kurzorových kláves šipka nahoru respektive šipka dolů. Aktivní (označené) mohou být nejen položky na stránce, ale i stránka samotná (název stránky v řádku stránek). Označit naopak nelze položky, které jsou pouze textové (nemají hodnotu). Podobně jako pohyb po stránkách je ovládán i pohyb po položkách. Směrem dolů je pohyb cyklický, tedy postupně prochází položky dolů (kromě čistě textových) a z poslední aktivní položky se přejde po dalším stisku klávesy na název stránky. Směrem nahoru se pohyb zastaví na názvu stránky.

Aktivní položka nebo stránka reaguje na stisk klávesy Enter následovně: pokud je aktivní název stránky a v definici stránky je uveden ukazatel na funkci stránky, pak se tato funkce spustí. Pokud aktivní položka na stránce obsahuje pouze ukazatel na zápisovou funkci, funkce se spustí a její výsledek se zobrazí do řádku popisu. Atributy písma zobrazení výsledné zprávy reflektují typ v návratové hodnotě stejně jako je tomu u hodnot položek na stránce. Chybový typ navíc pošle do terminálu příkaz pro zvukový signál (zvonek). Obsahuje-li aktivní položka ukazatele na zápisovou i čtecí funkci, je zahájen proces editace hodnoty dané položky. V každém z uvedených případů dojde při jejich provedení k aktualizaci všech hodnot položek na stránce.

Pokud aktivní položka obsahuje oba ukazatele na funkce (čtecí i zápisovou), reaguje navíc i na stisk klávesy Delete. V takovém případě dojde ke spuštění zápisové funkce

s prázdným ukazatelem (NULL) na novou hodnotu položky. Tuto funkcionalitu lze využít k rychlému smazání hodnot bez nutnosti jejich editace.

Varianta knihovny BIOS musí mít, na rozdíl od varianty MENU, možnost ukončení svého běhu. Tuto funkcionalitu může zajistit více než jedna položka v menu a to tím, že je typ v návratové hodnotě zápisové funkce nastaven na BiosExit. Po spuštění a návratu z takové zápisové funkce zajistí knihovna ve variantě BIOS výpis návratové zprávy a ukončí se. Ve variantě BIOS lze ještě kdykoli, bez ohledu na aktivní stránku a položku, ukončit běh knihovny stiskem klávesy Break.

4.5.2 Ovládání při editaci hodnot

Editace hodnoty po jejím spuštění zobrazí na začátek editačního řádku prompt, za ním původní hodnotu položky a kurzor nastaví za poslední znak původní hodnoty. Po řádku lze pohybovat kurzorem (vlevo, vpravo) v rozmezí od prvního znaku až na pozici za poslední znak. Stiskem klávesy Backspace se maže znak vlevo od kurzoru. Klávesa Delete maže znak na pozici kurzoru, případně znak vlevo od kurzoru, pokud je kurzor za pozicí posledního znaku hodnoty. Běžné znakové klávesy (ASCII 0x20-0x7E) zapíší znak na pozici kurzoru a zbytek textu posunou. Délka textu editované hodnoty je omezena podle velikosti displeje (ve znacích) po odečtení délky promptu a nutných mezer před začátkem a za koncem upravovaného textu. Po naplnění maximální délky pak přidávání dalších znaků není možné (na stisk kláves editor nereaguje).

Pro potvrzení nově zadané hodnoty a ukončení editace je potřebné stisknout klávesu Enter. Přerušení editace (bez změny hodnoty) lze provést stiskem klávesy Escape. Ve variantě MENU je pro přerušení editace nutný dvojitý stisk klávesy Escape, neboť tato varianta nehlídá časovou návaznost řídících kódů přicházejících z terminálu.

5 Návrh a realizace prototypové aplikace

Požadavky pro vývoj prototypové aplikace a zařízení, do kterých bude knihovna TUI integrována, jsem si stanovil takto:

- aplikace by měla demonstrovat všechny vlastnosti navrhované knihovny a být schopna provozu na mikrokontrolérech od výrobců Atmel i Microchip
- mikrokontrolér by měl mít co nejslabší parametry, ale postačovat pro provoz aplikace včetně integrované knihovny
- k vybranému mikrokontroléru musí existovat vývojový kit, zajišťující alespoň minimální podporu při vývoji a testování
- aplikace by se měla na obou platformách chovat identicky, měla by mít stejným způsobem integrovanou knihovnou a stejně definované menu

Jako mikrokontrolér od společnosti Microchip jsem se rozhodl využít PICKit Demo Board 44-pin, osazený mikrokontrolérem PIC18F45K20 a připojený přes PICKit 3 programátor respektive debugger. Jako vývojové prostředí pro prototypovou aplikaci s integrovanou knihovnou pro Mikrochip mikrokontrolér jsem použil MPLAB X IDE ve verzi 2.20.

Pro mikrokontrolér z produkce Atmel jsem využil testovací modul, využívaný při výuce APPS na katedře informatiky, osazený čipem AT-Mega32. Jako vývojové prostředí pro tento typ mikrokontroléru jsem použil AtmelStudio ve verzi 6.2.

Po volbě obou mikrokontrolérů s jejich vývojovými deskami bylo podstatné najít co nejvíce společných ovládacích a zobrazovacích prvků v jejich zapojení. Obě vývojové desky jsou osazeny poměrně malým počtem tlačítek a LED diod. Nabízelo se tak řešení omezit aplikaci na tyto společné prvky nebo rozšířit zapojení tak, aby desky obsahovaly více společných prvků.

Pro napojení obou platforem k terminálu jsem pro vývoj i testování prototypové aplikace s integrovanou knihovnou využil sériové linky. Deska zapůjčená z katedry již tuto možnost komunikace zahrnuje simulovaně přes USB port. Pro vývojovou desku PICKit jsem využil připojení převodníku RS232 na USB. Jako terminál jsem pro vývoj využil terminálovou aplikaci Hyperterminal na platformě MS Windows [10].

Napadlo mně využít jako základní myšlenku pro vytvoření aplikace můj historický generátor náhodných kódů Morseovy abecedy, kdysi určený pro nácvik příjmů telegramových pětimístných šifer. Ani jedna z desek sice neobsahovala reproduktor, měnič či jinou součástku vydávající zvuk, ale LED diody jsou pro tento účel dostatečné (blikající světlo i v dnešní době vojenským námořníkům při rádiovém klidu ke komunikaci stačí).

5.1 Návrh prototypové aplikace

Převést původní myšlenku do realizovatelné aplikace pracující identicky na obou platformách nebylo tak složité. Jenom jsem se pro zjednodušení zbavil prvku náhodného generování a nahradil jej pevně zadaným textem. Navrhl jsem aplikaci tak, aby mohla fungovat samostatně i bez integrace s připravenou knihovnou TUI, s využitím pevně nastavených vlastností a ovládaná tlačítky na desce. Integrace s knihovnou pak podstatně rozšiřuje možnosti nastavení vlastností a ovládání aplikace, včetně schopnosti zobrazování aktuálního stavu.

Aplikace umí samostatně odvysílat dva nastavené texty v Morseově kódu bez knihovny TUI. U obou z nich se dá nastavit rychlost vysílání, konkrétně délka bodu (jedné tečky) a počet opakování textu. Vysílání probíhá pomocí LED diod v nastaveném režimu. Integrovaná knihovna nejenže umožní nastavit a za běhu měnit vysílané texty, rychlost i počet opakování, ale také umožňuje změnit režim nastavení LED diod pro vysílání. Ve variantě MENU umožní knihovna i přímé řízení zahájení a ukončení vysílání nastavených textů.

Ačkoliv je funkcionalita knihovny demonstrovaná na jednoduché aplikaci, jsou zde využité všechny její vlastnosti a možnosti. Využití různých datových typů a využití jakékoliv spustitelné funkce v rámci vytvářených aplikací, které budou knihovnu integrovat, je omezeno pouze elektrotechnickým zapojením zařízení a fantazií i schopnostmi vývojáře.

5.2 Návrh a vytvoření menu, stránek a jejich položek

K rozložení zobrazovaných informací a funkcí aplikace na plochu displeje se velmi hodí navržená knihovna TUI a její struktura stránek a položek na nich. Stránky i bloky položek mohou být tematicky nebo funkčně společné, což zvyšuje přehlednost zobrazených informací a intuitivnost ovládání.

Příklady zdrojových kódu i snímků displeje terminálu uváděné v této kapitole jsou z prototypové aplikace, připravené pro mikrokontrolér Atmel ATmega32 na vývojové desce AVR-Kit.

5.2.1 Vzhled stránek a jejich položek

V prototypové aplikaci jsem se rozhodl v menu využít tří stránek, kde na první z nich jsou nastavitelné vlastnosti vysílaných textů, na druhé pak vlastnosti zobrazení vysílání. Na poslední třetí stránce je funkční ovládání aplikace, které se ve zdrojovém kódu modifikuje podle zvolené varianty implementace knihovny (BIOS/MENU).

V první řadě je potřebné vytvořit a naplnit instanci struktury (viz kód 5.1) definující samotné menu (definice struktury menu viz: 4.2.1). Titulek menu může být i víceřádkový, v takovém případě je však potřeba počítat s posunem ostatních řádků. Titulek menu se zobrazuje do prvního řádku, všechny další jsou pak v instanci struktury menu uvedeny. Ukazatele na povinné funkce rozhraní odkazují na funkce dodané v samostatných souborech podle konkrétního výrobce a mikrokontroléru.

```
const ucMenu menu = {                                                    (5.1)
    (char*) "Testovaci menu pro ucBios/ucMenu v1.0, pripraveno pro testovaci desku AVR-Kit",
    menuPages, MenuInitialization, MenuDispose,
    3, 5, 16, 18, 50, (char*)"/:>",
    ReceiveCountAVR, ReceiveCharAVR, TransmitCharAVR, IntToDecStrAVR,
    RefreshCounterReadAVR, RefreshCounterResetAVR, MenuSleepAVR };
```

Instance struktury menu v sobě ještě obsahuje ukazatel na pole se seznamem stránek. Každá ze stránek v seznamu obsahuje název, hodnotu čítače pro aktualizaci, seznam položek na stránce a ukazatel na vlastní zápisovou funkci stránky (viz 5.2).

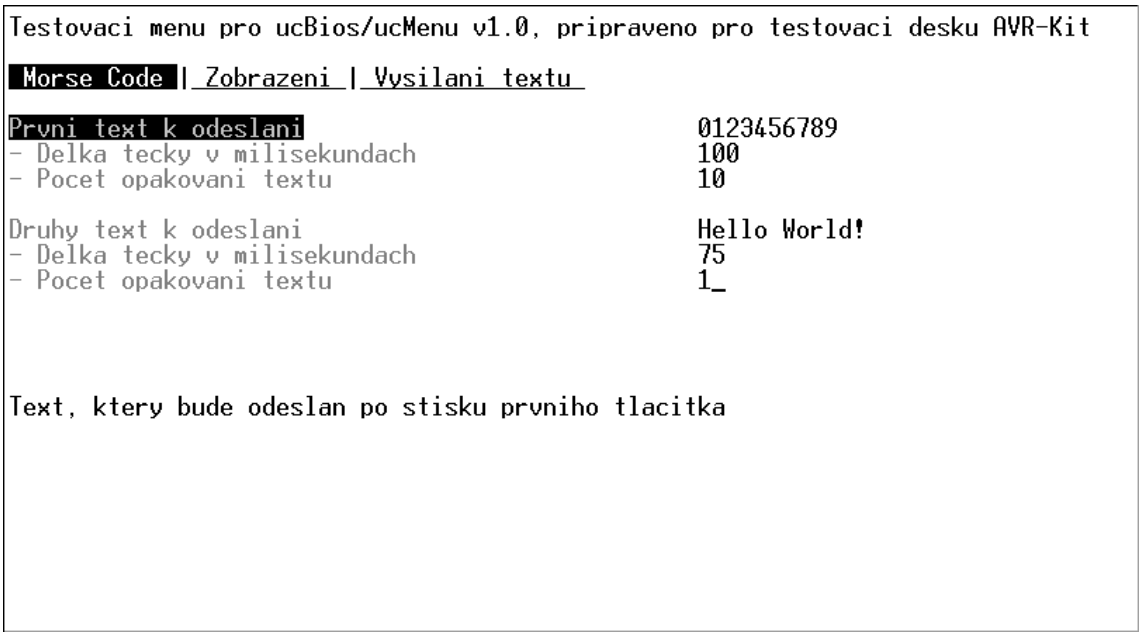
Názvy stránek by měly být jen tak dlouhé, aby text složený ze všech názvů včetně oddělovačů (3 znaky) nepřesáhl šířku displeje terminálu. Názvy stránek za sebou lze stejně jako titulek menu zobrazit na více řádcích. Je však potřebné upravit názvy tak, aby jejich zalomení na konci řádku bylo estetické a je potřeba myslet i na posunutí řádků v definici menu. Čítač pro aktualizaci může mít hodnotu 0, pokud stránka aktualizaci nepožaduje (v mém případě 2. stránka), jinak je to hodnota, po jejímž dosažení knihovna zajistí automatickou aktualizaci

hodnot položek na stránce. Každá stránka může obsahovat ukazatel na pole se seznamem položek na stránce. Posledním prvkem ve struktuře popisující stránku je ukazatel na zápisovou funkci stránky. Pole se seznamem stránek musí být ukončeno posledním prvkem s hodnotou NULL. Tímto prvkem knihovna rozezná konec pole, podobně jako konec řetězce znakem 0x00.

```
const ucMenuPage menuPages[] = {                                    (5.2)
    { (char*)"Morse Code", 200, menuItems1, NULL },
    { (char*)"Zobrazeni", 0, menuItems2, NULL },
    { (char*)"Vysilani textu", 20, menuItems3, item30Write },
    { NULL } };
```

Pole se seznamem položek pro kteroukoliv stránku obsahuje instance struktur položek ukončené posledním prvkem s hodnotou NULL. Zvláštní pozornost je nutné věnovat položce, která má na displeji reprezentovat prázdný řádek. Prázdný řádek v sobě musí obsahovat alespoň jednu mezeru v názvu, jinak by byl chybně považován za konec celého pole položek na stránce.

```
const ucMenuItem menuItems1[] = {                                   (5.3)
    { (char*)"Prvni text k odeslani",
      (char*)"Text, ktery bude odeslan po stisku prvniho tlacitka",
      item11Read, item11Write },
    { (char*)"- Delka tecky v milisekundach",
      (char*)"Cas trvani jedne tecky v milisekundach v intervalu 50-1000ms",
      item12Read, item12Write },
    { (char*)"- Pocet opakovani textu",
      (char*)"Pocet opakovani vysilani textu po stisku tlacitka 0-1000",
      item13Read, item13Write },
    { (char*)" ",
      (char*)"",
      NULL, NULL },  //Prazdny radek (musi byt alespon jedna mezera v nazvu)
    { (char*)"Druhy text k odeslani",
      (char*)"Text, ktery bude odeslan po stisku druheho tlacitka",
      item14Read, item14Write },
    { (char*)"- Delka tecky v milisekundach",
      (char*)"Cas trvani jedne tecky v milisekundach v intervalu 50-1000ms",
      item15Read, item15Write },
    { (char*)"- Pocet opakovani textu",
      (char*)"Pocet opakovani vysilani textu po stisku tlacitka 0-1000",
      item16Read, item16Write },
    { NULL } };
```

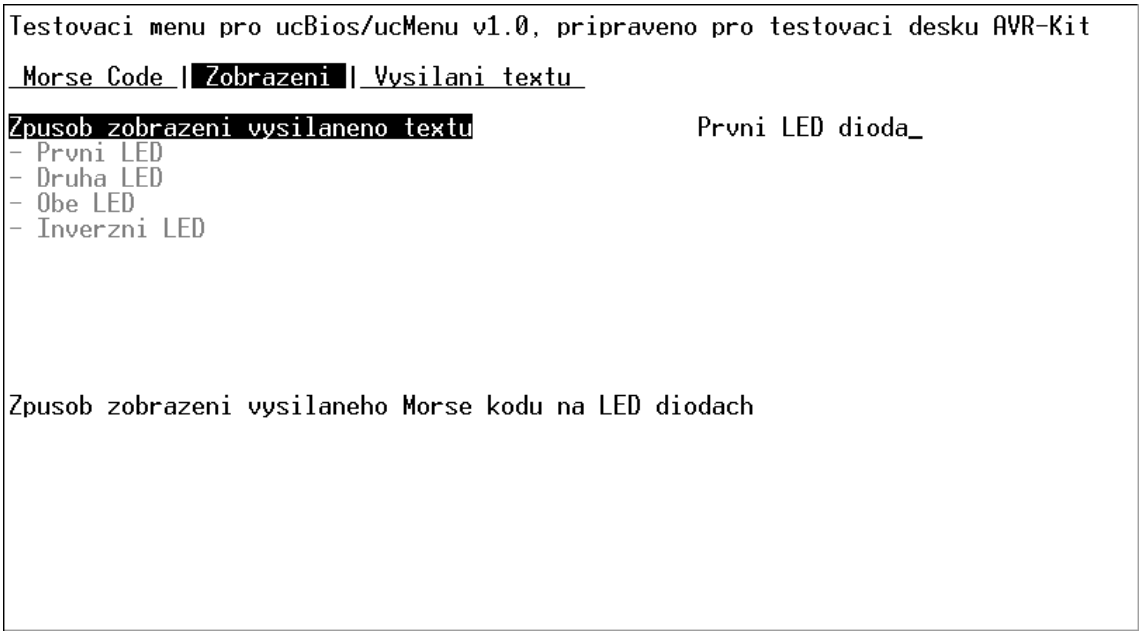


Obrázek 5.1: *První stránka menu prototypové aplikace*

První stránka v prototypové aplikaci obsahuje dva na pohled podobné bloky vlastností vysílaných textů. Aplikace umí vysílat dva odlišné texty různou rychlostí a v různém počtu opakování. Ve fragmentu zdrojového kódu (5.3) i snímku displeje (obrázek 5.1) z prototypové aplikace je vidět, jak se struktura definující stránku následně graficky zobrazí. Na první stránce lze pro každý z textů zadat jeho obsah v rozsahu maximálně 50 znaků. Rychlost vysílání je dána rozměrem jedné tečky a její délka se zadává v milisekundách. Jen připomenu pravidla Morseova kódu, kdy čárka má délku tří teček, mezery mezi čárkami a tečkami ve znaku mají délku jedné tečky, mezera mezi znaky trvá jako pět teček. Lze pak dovodit, že délka mezery mezi slovy a na konci věty je stejná jako dvě za sebou jdoucí mezery mezi znaky, tedy v trvání deseti teček. Rychlost vysílání je tedy relativní podle vysílaných znaků v textu a uvádí se proto jen délka jedné tečky. Posledním údajem v bloku obou textů je počet opakování vysílání celého textu. Pokud se v prototypové aplikaci zadá nula, bude se text ve vysílání opakovat donekonečna. První stránka obsahuje pouze položky obsahující oba ukazatele na zápisovou i čtecí funkci.

Na druhé stránce v prototypové aplikaci jsou demonstrovány možnosti knihovny pracovat s položkami obsahujícími pouze jeden z ukazatelů na funkci (čtecí nebo zápisovou). Příklad zdrojového kódů (5.4) a snímek displeje (obrázek 5.2) znázorňují jejich souvislost.

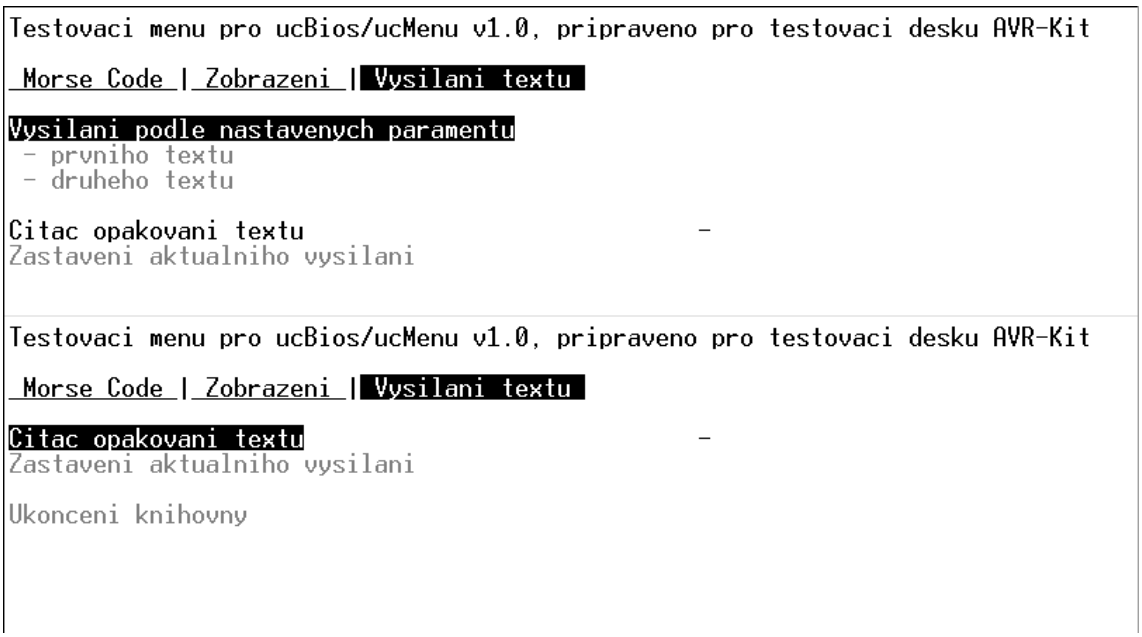
```
const ucMenuItem menuItems2[] = {                                                    (5.4)
    { (char*)"Zpusob zobrazeni vysilaneno textu",
      (char*)"Zpusob zobrazeni vysilaneho Morse kodu na LED diodach",
      item21Read, NULL },
    { (char*)"- Prvni LED",
      (char*)"Text bude vysilan (zobrazen) pouze na prvni LED",
      NULL, item22Write },
    { (char*)"- Druha LED",
      (char*)"Text bude vysilan (zobrazen) pouze na druhe LED",
      NULL, item23Write },
    { (char*)"- Obe LED",
      (char*)"Text bude vysilan (zobrazen) na obou LED soucasne",
      NULL, item24Write },
    { (char*)"- Inverzni LED",
      (char*)"Text bude vysilan (zobrazen) na prvni LED normalne a na druhe invertovane",
      NULL, item25Write },
    { NULL } };
```



Obrázek 5.2: *Druhá stránka menu prototypové aplikace*

Druhá stránka obsahuje nejdříve položku pouze s ukazatelem na čtecí funkci, která navíc nepracuje se základním datovým typem, ale s výčtem. Položka reprezentuje hodnotu pro formu vysílání Morseova kódu na LED diodách. Další čtyři položky obsahují pouze ukazatele na zápisové funkce a tím demonstrují položky pro volání funkce. Pokud se taková aktivní položka potvrdí, nenastane žádná editace hodnoty, ale pouze se spustí nastavená funkce a po jejím ukončení se zobrazí výsledná zpráva. V prototypové aplikaci tyto čtyři položky volají samostatné funkce, ze kterých každá nastavuje jednu z forem vysílání.

```
const ucMenuItem menuItems3[] = {                                    (5.5)
#ifndef UC_TUI_AS_BIOS_NOT_MENU
    { (char*)"Vysilani podle nastavenych paramentu",
      (char*)"",
      NULL, NULL },  //Prazdny radek (musi byt alespon jedna mezera v nazvu)
    { (char*)" - prvniho textu",
      (char*)"Zastavi aktualni vysilani pokud nejake probiha a zacne vysilat prvni text",
      NULL, item31Write },
    { (char*)" - druheho textu",
      (char*)"Zastavi aktualni vysilani pokud nejake probiha a zacne vysilat druhy text",
      NULL, item32Write },
    { (char*)" ",
      (char*)"",
      NULL, NULL },  //Prazdny radek (musi byt alespon jedna mezera v nazvu)
#endif
    { (char*)"Citac opakovani textu",
      (char*)"",
      item33Read, NULL },
    { (char*)"Zastaveni aktualniho vysilani",
      (char*)"Zastavi aktualni vysilani",
      NULL, item34Write },
#ifdef UC_TUI_AS_BIOS_NOT_MENU
    { (char*)" ",
      (char*)"",
      NULL, NULL },  //Prazdny radek (musi byt alespon jedna mezera v nazvu)
    { (char*)"Ukonceni knihovny",
      (char*)"(funkcni v BIOS variante)",
      NULL, item35Write },
#endif
    { NULL } };
```



Obrázek 5.3: *Třetí stránka menu prototypové aplikace (horní část MENU, dolní část BIOS)*

Příklad fragmentu zdrojového kódu (5.5) a snímku displeje (obrázek 5.3) demonstrují definici třetí strany a dva odlišné způsoby jejího výsledného zobrazení závislé na parametru určujícím vybranou implementaci knihovny (BIOS nebo MENU).

Pro verzi BIOS je na třetí stránce demonstrována jedna z funkcí (Ukončení knihovny) vracející v návratové hodnotě typ BiosExit. Tato funkce ukončí činnost knihovny a předá řízení zpět do místa, odkud byla knihovna původně spuštěna (v prototypové aplikaci je to hlavní vlákno aplikace).

Pro verzi MENU jsou na třetí stránce funkce vysílání textů, které v prototypové aplikaci demonstrují výhodu této varianty implementace knihovny. Obě zmíněné funkce ukončí případně běžící vysílání a nově zahájí vysílání textu podle jeho aktuálně nastavených vlastností.

Obě varianty třetí strany menu obsahují položku s pouze jedním ukazatelem na čtecí funkci zobrazující hodnotu čítače opakování aktuálně vysílaného textu.

5.2.2 Funkce stránek a položek na stránkách

Jak už bylo v předchozích kapitolách popsáno, vracejí čtecí i zápisové funkce shodný návratový typ (strukturu *ucResponse*). Pro co největší úsporu paměti i pro snadnou implementaci jsem v prototypové aplikaci hned z počátku deklaroval a naplnil standardní návratové hodnoty (*ResponseOK, ResponseSVOK, ResponseCBOK*). Mezi ně pak patří i deklarace předávané návratové hodnoty, která se od standardních odlišuje (*response*), (kód 5.6).

```
ucResponse ResponseOK   = { Information, NULL };                                    (5.6)
ucResponse response;
ucResponse ResponseSVOK = { Information, (char*)"Nove zadana hodnota byla v poradku ulozena." };
ucResponse ResponseCBOK = { Information, (char*)"Volani funkce probehlo v poradku." };
```

Definice stránky může obsahovat i ukazatel na zápisovou funkci. V prototypové aplikaci je taková funkce demonstrována na třetí stránce. Tato konkrétní funkce má za úkol zastavit probíhající vysílání (viz kód 5.7).

```
ucResponse item30Write(char *value) {                                              (5.7)
    StopSending();
    return ResponseCBOK; }
```

Podrobnější popis funkcí položek na stránce menu jsem pro přehlednost rozdělil na skupiny čtecích a zápisových funkcí. Následně uvedu jen příklady zdrojových kódů funkcí pro znázornění toho, jak jsou takové funkce implementované v prototypové aplikaci. Příklady mohou sloužit i jako návod pro budoucí implementaci čtecích i zápisových funkcí vývojářem při integraci knihovny TUI.

Všechny čtecí funkce by v textu návratové hodnoty měly uvést textovou reprezentaci hodnoty dané položky (vlastnosti v jakémkoliv datovém typu). Do typu v návratové hodnotě se uvádí prvek z výčtu (*ucResponseType {None, Information, Warning, Error}*), podle kterého bude textová reprezentace hodnoty zobrazena.

Příklad pro čtecí funkci vracející hodnotu položky (datového typu řetězec), (viz kód 5.8).

```
ucResponse item11Read(void) {                                                      (5.8)
    response.ResponseType = None;
    response.ResponseText = gcaFirstText;
    return response; }
```

Příklad čtecí funkce vracející hodnotu položky (datového typu integer), (viz kód 5.9).

```
ucResponse item12Read(void) {                                    (5.9)
    itoa(giFirstDotDuration, DecadicNumber, 10);
    response.ResponseType = None;
    response.ResponseText = DecadicNumber;
    return response; }
```

Další příklad, ve kterém je sice také čtena hodnota vlastnosti datového typu integer, ale v určitém stavu aplikace je v textu návratové hodnoty jiný než číselný obsah (pomlčka). Takto lze převádět, modifikovat nebo doplňovat textovou reprezentaci jakéhokoliv datového typu. Je možné za údaj doplnit jednotku, řád a podobně (viz kód 5.10).

```
ucResponse item33Read(void) {                                    (5.10)
    response.ResponseType = None;
    response.ResponseText = (char*)"-";
    if (gcSendingTextIndex == 0)
        return response;
    itoa(giSendingTextCount + 1, DecadicNumber, 10);
    response.ResponseText = DecadicNumber;
    return response; }
```

Posledním příkladem (5.11) čtecí funkce je překlad vlastnosti s výčtem do textové podoby.

```
ucResponse item21Read(void) {                                    (5.11)
    switch (geLedMode) {
        case FirstLed:
            response.ResponseText = (char*)"Prvni LED dioda";
            break;
        case SecondLed:
            response.ResponseText = (char*)"Druha LED dioda";
            break;
        case BothLeds:
            response.ResponseText = (char*)"Obe LED diody";
            break;
        case InvertLeds:
            response.ResponseText = (char*)"Obe LED v inverzi";
            break;
    }
    response.ResponseType = None;
    return response; }
```

Druhým blokem jsou funkce zápisové. Ty lze volat jako součást položek na stránce menu, které mají uvedený i ukazatel na čtecí funkci, pak se taková funkce volá po editaci nové hodnoty nebo při jejím odstranění. Pokud má položka na stránce menu uveden pouze ukazatel na zápisovou funkci, pak je tato zápisová funkce spuštěna při volbě dané položky.

Prvním příkladem (5.12) je funkce pro zápis hodnoty položky datového typu řetězec. V tomto případě se z editované hodnoty převezme nejvýše počet znaků odpovídající nastavenému maximálnímu počtu znaků pro vysílané texty bez ohledu na to, že při editaci hodnoty jich lze napsat více.

```
ucResponse item11Write(char *value) {                            (5.12)
    unsigned char i;
    for (i = 0; i < MORSE_TEXTS_LENGTH; i++) {
        gcaFirstText[i] = value[i];
        if (value[i] == 0) break;
    }
    gcaFirstText[MORSE_TEXTS_LENGTH - 1] = 0;
    return ResponseNope; }
```

V dalším příkladu (5.13) je uvedena zápisová funkce pro vlastnost datového typu integer. Tato funkce je doplněna o kontrolu rozsahu zadané hodnoty. V případě, kdy není rozsah dodržen,

vrací jinou návratovou hodnotu informující o chybě. Pokud je rozsah dodržen, pak funkce vrací předdefinovanou standardní odpověď.

```
ucResponse item12Write(char *value) {                                    (5.13)
    int iTemp;
    iTemp = atoi(value);
    if (iTemp < 50 || iTemp > 1000)     {
        response.ResponseType = Error;
        response.ResponseText = (char *)"Zadana hodnota neni v rozsahu 50..1000";
        return response; }
    giFirstDotDuration = iTemp;
    return ResponseSVOK; }
```

Příklad funkce pro nastavení jedné z hodnot výčtu do vlastnosti aplikace (čtený jinou funkcí v příkladu uvedeném výše). Funkce taktéž vrací předdefinovanou odpověď (viz kód 5.14).

```
ucResponse item22Write(char *value) {                                    (5.14)
    geLedMode = FirstLed;
    return ResponseSVOK; }
```

Příklad velmi jednoduché funkce (5.15), která je vždy proveditelná a tudíž může vrátit předem definovanou odpověď.

```
ucResponse item34Write(char *value) {                                    (5.15)
    StopSending();
    return ResponseCBOK; }
```

5.3 Povinné funkce rozhraní - ovladače

Knihovna ke své integraci do stávající či nové aplikace nějakého technického zařízení řízeného mikrokontrolérem potřebuje napojení na jeho prostředky, určené ke komunikaci i k časování a k nejednotným funkcím. Tato potřeba je dána rozdílnou konstrukcí použitého mikrokontroléru, vlastního zařízení a také rozdílností využívaného překladače jazyka C. Funkcí není mnoho, pro fungování knihovny TUI jsou ale nezbytné stejně tak jako jsou nezbytné ovladače v jakémkoliv jiném systému.

Společně s knihovnou jsem vytvořil i dvě rozhraní pro výrobce mikrokontrolérů Atmel a Mikrochip. Tato rozhraní obsahují všechny potřebné funkce pro rychlou integraci knihovny na mikrokontroléry obou výrobců s co nejmenším nárokem na úpravy. Základní podmínkou integrace je jen to, aby se aplikace společně s knihovnou do vybraného čipu paměťově vešla i po případných úpravách funkcí rozhraní, společně s podmínkou schopnosti zařízení nějakým způsobem komunikovat s terminálem. V připravených rozhraních je komunikace orientovaná na využití sériové komunikace. Pro bližší pochopení příkladů uvádím deklarace proměnných rozhraní (viz kód 5.16):

```
#define RECEIVE_BUFFER_LENGHT 64              //max 255                    (5.16)
Int  iRefreshCounter;
char caReceivedBuffer[RECEIVE_BUFFER_LENGHT];  //FIFO Buffer
int  iReceivedIndex = 0;
int  iReceivedCount = 0;
ucResponse ResponseNopeAVR = {None, NULL};
```

Povinné funkce pro inicializaci i pro ukončení knihovny stejně tak jako některé další funkce, které jsou úzce vázány na technologie mikrokontroléru, budou popsány v samostatných podkapitolách. Existují však takové, které jsou v obou rozhraních obsahem identické a liší se jen sufixem v názvu funkce identifikujícího výrobce (AVR pro Atmel a PIC pro Microchip). Komentáře, které jsou součástí zdrojových kódů, jsou v uvedených příkladech odstraněny.

5.3.1 Obsahově identické funkce obou platforem

Funkce zajišťující přečtení počtu znaků přijatých z terminálu a čekajících v zásobníku na další zpracování (*ReceiveCount**) je v rozhraních zpracovaná následovně (viz kód 5.17):

```
int ReceiveCountAVR(void) {                                                    (5.17)
    iRefreshCounter++;
    return iReceivedCount; }
```

Funkce zajišťující přečtení jednoho znaku (nejdříve přijatého z terminálu) ze zásobníku (*ReceiveChar**) je zpracovaná následovně (viz kód 5.18):

```
unsigned char ReceiveCharAVR(void) {                                           (5.18)
    signed int iFirstIndex;
    if (iReceivedCount == 0) return '\0';
    iFirstIndex = iReceivedIndex - iReceivedCount;
    if (iFirstIndex < 0) iFirstIndex = iFirstIndex + RECEIVE_BUFFER_LENGHT;
    iReceivedCount--;
    return caReceivedBuffer[iFirstIndex]; }
```

Funkce pro přečtení hodnoty čítače určeného k automatické aktualizaci zobrazení hodnot položek na stránce menu (*RefreshCounterRead**) je zpracovaná takto (viz kód 5.19):

```
int RefreshCounterReadAVR(void) {                                              (5.19)
    return iRefreshCounter; }
```

Poslední identická funkce pro vynulování čítače pro automatickou aktualizaci zobrazení (*RefreshCounterReset**) je v obou rozhraních zpracovaná takto (viz kód 5.20):

```
void RefreshCounterResetAVR(void) {                                            (5.20)
    iRefreshCounter = 0; }
```

5.3.2 Obsahově odlišné funkce obou platforem

Následně popíšu funkce, které jsou fixovány na použitý mikrokontrolér, vývojovou desku i na využitý překladač. Při integraci knihovny do aplikace v jiném zařízení bude s největší pravděpodobností nutné tyto funkce upravit tak, aby reflektovaly možnosti nově zvoleného mikrokontroléru i elektrotechnického zapojení zařízení. U každé funkce uvádím oba způsoby implementace jak pro Atmel tak i pro Microchip. Lze tak dobře srovnat rozdílnost přístupu obou výrobců a rozdílnost programového řešení identické funkce u obou platforem.

Začnu důležitou funkcí (*MenuInitialization**), kterou knihovna využívá při své inicializaci. Tato funkce by měla obsahovat nastavení i inicializaci komunikačního rozhraní s terminálem a také nastavení a povolení přerušení, které se o tuto komunikaci postará. V připravených rozhraních je funkce pro komunikaci přes USART vytvořená následovně (viz kód 5.21):

```
ucResponse MenuInitializationPIC(void) {                                       (5.21)
    TRISCbits.TRISC7=1;
    TRISCbits.TRISC6=0;
    UART1Config = USART_TX_INT_OFF & USART_RX_INT_OFF &
                  USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_BRGH_HIGH ;
    baud = 51;
    OpenUSART(UART1Config, baud);
    RCIF = 0;    //reset RX pin flag
    RCIP = 0;    //Not high priority
    RCIE = 1;    //Enable RX interrupt
    PEIE = 1;    //Enable pheripheral interrupt (serial port is a pheripheral)
    ei();        //master switch for interrupt?
    MenuSleepPIC(1000);
    putsUSART(mi);
    MenuSleepPIC(1000);
    return ResponseNopePIC; }
```

```
ucResponse MenuInitializationAVR(void) {
    UCSRA |= _BV(U2X);
    UBRRL = ( ( float ) F_CPU / (8UL * USART_BAUDRATE)) + 0.5 - 1;
    UCSRB = ((1<<TXEN)|(1<<RXEN) | (1<<RXCIE));
    sei();
    return ResponseNopeAVR; }
```

Rozhraní jsou doplněna o funkci zpracovávající přerušení při příjmu znaků přes sériovou linku USART. Pokud aplikace využívá i jiná přerušení, je potřebné na to myslet a zpracovat všechna využívaná přerušení. Prototypová aplikace zachytává pouze přerušení pro příjem USART a v připravených rozhraních jsou zpracovaná takto (viz kód 5.22, první PIC následované AVR):

```
void interrupt PicInterrupt() {
    InterruptPIC(); }

void InterruptPIC() {
    //check if the interrupt is caused by RX pin
    if(PIR1bits.RCIF == 1 && iReceivedCount < RECEIVE_BUFFER_LENGHT) {
        caReceivedBuffer[iReceivedIndex] = ReadUSART();
        iReceivedIndex++;
        iReceivedCount++;
        if (iReceivedIndex >= RECEIVE_BUFFER_LENGHT) iReceivedIndex = 0;
        PIR1bits.RCIF = 0; // clear rx flag
    } }


ISR(USART_RXC_vect) {
    if (iReceivedCount < RECEIVE_BUFFER_LENGHT) {
        caReceivedBuffer[iReceivedIndex] = UDR;    //Read Character from UART reg.
        iReceivedIndex++;
        iReceivedCount++;
        if (iReceivedIndex >= RECEIVE_BUFFER_LENGHT) iReceivedIndex = 0;
    } }
```

(5.22)

Funkce *MenuSleep** na pozdržení vlákna procesu na přesně stanovenou dobu je využívaná jenom ve variantě BIOS implementace knihovny. Je určená ke zpoždění při zpracování příchozích znaků a tím k rozpoznání řídících sekvencí terminálů od samotného stisku klávesy Escape. Ve vytvořené prototypové aplikaci je funkce využita navíc i k časování vysílání textů v Morseově abecedě. V rozhraních je funkce zpracovaná takto (viz 5.23):

```
void MenuSleepPIC(int miliseconds) {
    unsigned char m256 = ms >> 8;
    unsigned char m16 = ( ms & 0xFF ) >> 4;
    char m = ms & 0x0f;
    while ( m256-- ) _delay( 256 );
    while ( m16-- ) _delay( 16 );
    while ( m-- ) _delay( 1 );
}

void MenuSleepAVR(int miliseconds) {
    unsigned char m256 = ms >> 8;
    unsigned char m16 = ( ms & 0xFF ) >> 4;
    char m = ms & 0x0f;
    while ( m256-- ) _delay_ms( 256 );
    while ( m16-- ) _delay_ms( 16 );
    while ( m-- ) _delay_ms( 1 );
}
```

(5.23)

Důležitá je i funkce *TransmitChar** pro odeslání znaku do připojeného terminálu. V obou vytvořených rozhraních jsou funkce zpracované tak, že k odesílání znaků nevyužívají zásobník a přerušení, ale čekají na uvolnění vysílače USART a teprve pak znak odešlou. Tato varianta je méně náročná na datovou paměť, ale na druhou stranu může aplikaci zdržovat a to hlavně ve variantě MENU implementace knihovny. Zpracování funkcí v rozhraních viz kód 5.24.

```
void TransmitCharPIC(unsigned char chr) {                    (5.24)
    while (BusyUSART());
    putcUSART(chr); }


void TransmitCharAVR(unsigned char chr) {
    while((UCSRA &(1<<UDRE)) == 0);
    UDR = chr; }
```

Rozdílnost překladačů a v nich implementované funkce pro překlad čísla do textového řetězce v dekadické podobě si vynutily doplnění povinných funkcí o další funkci *IntToDecStr**. V obou rozhraních je tak vytvořen jen nepatrně odlišný kód (prohozené argumenty viz kód 5.25):

```
void IntToDecStrPIC(int value, char *text) {                 (5.25)
    itoa(text, value, 10); }


void IntToDecStrAVR(int value, char *text) {
    itoa(value, text, 10); }
```

Poslední funkcí vytvořených rozhraní je funkce pro ukončení (destruktor) knihovny. Využívá ji výhradně varianta implementace knihovny BIOS. Tato funkce má za úkol ukončit komunikaci s teminálem. V obou rozhraních jsou funkce provedeny následovně (viz 5.26):

```
ucResponse MenuDisposePIC(void) {                            (5.26)
     CloseUSART();
     return ResponseNopePIC; }


ucResponse MenuDisposeAVR(void) {
    UCSRB = ((0<<TXEN)|(0<<RXEN) | (0<<RXCIE));
    return ResponseNopeAVR; }
```

5.4 Integrace TUI do prototypové aplikace

Integrace připravené knihovny do prototypové aplikace na obou platformách je po provedení kroků z předchozích podkapitol velice snadná u obou platforem jak Atmel tak i Michrochip.

Do projektu s prototypovou aplikací jsem připojil soubory obou variant implementací knihovny *ucBios.h* + *ucBios.c* a *ucMenu.h* + *ucMenu.c*. Vývojář může připojit jen soubory pro jednu vybranou variantu implementace. Současně jsem k projektu připojil i soubory s obecnou částí knihovny *ucCommon.h* + *ucCommon.c*. Poslední do projektu připojené soubory jsou soubory rozhraní, které jsou závislé na platformě. Do prototypové aplikace pro výrobce Atmel jsem připojil připravené soubory rozhraní AVR *ucLinkAVR.h* + *ucLinkAVR.c*. Pro výrobce Microchip jsem připojil soubory rozhraní PIC *ucLinkPIC.h* + *ucLinkPIC.c*. Pokud bude vývojář integrovat knihovnu TUI a rozhodne se nevyužít připravená rozhraní, pak se musí postarat o povinné funkce rozhraní sám a zajistit i jejich připojení do projektu a vazby.

V souboru prototypové aplikace *ucMorseCodeMain.c* (kód 5.27) jsem v úvodu pro snadné předvedení obou variant implementace knihovny (BIOS i MENU) definoval konstantu:

```
#define UC_TUI_AS_BIOS_NOT_MENU                              (5.27)
```

Ponechání definice nebo její zakomentováním mi umožňuje jednoduše zvolit jednu z variant implementace a demonstrovat tak obě varianty bez složitějších změn zdrojového kódu. Tato konstanta je využita i v souborech rozhraní právě k tomu, aby byla připojena jen vybraná

varianta implementace knihovny. Připojení připraveného rozhraní do prototypové aplikace jsem podle použité platformy provedl následovně (viz kód 5.28):

```
#include "ucLinkAVR.h"        /        #include "ucLinkPIC.h"                    (5.28)
```

Připojené rozhraní si ve svém hlavičkovém souboru připojuje správnou variantu knihovny na základě výše popsané konstanty (*ucBios.h* nebo *ucMenu.h*). Obě varianty implementace knihovny si ve svém hlavičkovém souboru připojí obecnou část (*ucCommon.h* viz obr.: 4.1).

Do souboru prototypové aplikace jsem vložil připravenou definici menu, stránek i položek na stránkách a také všechny čtecí i zápisové funkce stránek a položek na stránkách.

Poslední úpravou prototypové aplikace, kterou jsem provedl v rámci její integrace s knihovnou TUI, je změna funkce *main* tak, aby aplikace spolupracovala s již připojenou knihovnou a definovaným menu. Toto připojení je rozdílné podle varianty implementace knihovny. Rozdíl v pojetí obou variant je popsaný v kapitole 4.3.2 a kapitole 4.3.3, včetně schematického znázornění.

V prototypové aplikaci jsem zvolil realizaci obou variant tak, že se na základě definované (nebo nedefinované) konstanty (*UC_TUI_AS_BIOS_NOT_MENU*) přeloží aplikace pro jednu zvolenou variantu implementace knihovny.

Principiální zdrojový kód funkce *main* (5.29) pro variantu implementace knihovny BIOS:

```
int main() {                                                                    (5.29)
    // inicializace aplikace
    while ( 1 ) {
        // opakovaný kod aplikace
        if (/*<<tlacitko>>*/) {
            RunBios(&menu, 0, 0); }  //Start TUI jako BIOS
    } }
```

Principiální zdrojový kód funkce *main* (5.30) pro variantu implementace knihovny MENU:

```
int main() {                                                                    (5.30)
    // inicializace aplikace
    menu.Initialization(); // inicializace TUI
    while ( 1 ) {
        // opakovaný kod aplikace
        DoMenuActions(&menu);  //Vykonej akci v TUI jako MENU
    } }
```

Tímto posledním krokem je knihovna TUI plně integrována do prototypové aplikace.

Závěr

Tato bakalářské práce vznikla na základě potřeby obecné knihovny pro textové uživatelské rozhraní, zajišťující komunikaci mezi zařízením, řízeným mikrokontrolérem, a textovým terminálem. Tato potřeba byla naplněna realizací knihovny TUI a jejích rozhraní. Knihovna umožňuje řízení aplikace v zařízení pomocí přehledného a intuitivního ovládání v textovém režimu. Řízení aplikace s využitím knihovny umožňuje zobrazení i nastavení hodnot vlastností a spouštění funkcí aplikace v zařízení.

Realizovaná knihovna a její rozhraní splňují požadavky, které jsem si stanovil ve fázi návrhu. Knihovna není závislá na způsobu připojení terminálu ani na komunikačním protokolu mezi terminálem a zařízením. Knihovna využívá jenom pět řídících sekvencí, které jsou nutné k obsluze terminálu. Knihovna zajišťuje zobrazení textové reprezentace jakéhokoliv datového typu. Knihovna automaticky aktualizuje zobrazované hodnoty položek na stránce v nastaveném časovém intervalu. Součástí knihovny je řádkový textový editor využívaný při zadávání a úpravě hodnot vlastností aplikace.

Pro usnadnění a urychlení integrace jsem vytvořil dvě varianty vzorového rozhraní. Ve vytvořených vzorových rozhraních je komunikace s terminálem prováděna přes rozhraní USART.

Knihovnu TUI lze implementovat do stávajících případně nově vznikajících aplikací s co nejmenším dopadem na existující kód aplikace. Vzhledem k využití jazyka C je knihovna přeložitelná pro různé druhy mikrokontrolérů od hlavních výrobců Microchip i Atmel.

Veškerou funkcionalitu knihovny TUI jsem demonstroval na prototypové aplikaci realizované s mírnou úpravou zdrojového kódu pro obě platformy výrobců (Atmel i Microchip). Prototypová aplikace v sobě obsahuje možnost jednoduše vybrat a zkompilovat zvolenou variantu implementací knihovny TUI (BIOS nebo MENU).

Projekty obsahující obě dvě varianty implementace knihovny TUI pro obě platformy (Microchip i Atmel) integrované do prototypové aplikace jsou uloženy na CD přiloženém k této bakalářské práci.

Zhodnocení navrženého rozhraní

Knihovna TUI z pohledu náročnosti plní požadovanou funkcionalitu pro obě řešené platformy se stejným zdrojovým kódem při využití rozhraní pro danou platformu. Výsledná kompilace kódu obsadí poměrně málo programové paměti mikrokontroléru a ponechává tak prostor pro hlavní aplikaci. Pro obě platformy se navíc nevytvářela jedna, ale dvě varianty řešení knihovny TUI. Na obou platformách tedy lze využít jak BIOS, tak i MENU variantu knihovny. Tato variabilita umožňuje budoucím vývojářům implementujícím realizovanou knihovnu vybrat si pro své řešení tu nejvhodnější variantu a to s využitím prakticky stejného kódu.

Stabilita knihovny TUI je velmi závislá na aplikaci, do které je integrována. Knihovna je jen podpůrný prostředek pro komunikaci mezi uživatelem a aplikací technického zařízení, ve svých funkcích je odladěna a otestována. Je však nutné v aplikaci myslet na to, že knihovna umožňuje měnit jednotlivé vlastnosti aplikace postupně, nikoliv najednou. Postupná změna vlastností může být žádoucí, ale může také způsobit nepředvídatelné stavy v hlavní aplikaci. Přesto i v tomto nabízí realizovaná knihovna možná řešení. Například inicializace a destrukce

menu může obsahovat změnu stavu, na který bude hlavní aplikace reagovat, případně využití funkcí pro uložení více nastavených hodnot do produkční části vlastností aplikace.

Integrace knihovny do stávající nebo nově vytvářené aplikace není příliš složitá. Programově záleží složitost jen na rozsahu změn v realizovaném rozhraní nebo v jeho nové implementaci. Definice menu, stránek a položek na stránkách je triviální a vývojář si snadno pomůže kopírováním podobně jako u funkcí k jednotlivým položkám. Složitost definiční části záleží jen na velikosti a struktuře vývojářem navrhovaného menu. Projekty na přiloženém CD poskytují budoucím vývojářům nejen knihovnu a její rozhraní, ale i možnost využít prototypovou aplikaci s již integrovanou knihovnou pro své vlastní projekty. Obecně za celou knihovnu lze shrnout, že integrace by neměla dělat problém ani začínajícím vývojářům.

Vzhledem k tomu, že jsem knihovny vytvářel pro dvě různé platformy a ve dvou variantách implementace, je zajímavé porovnání výsledného využití paměti pro všechny čtyři případy. Výsledky obsazení paměti (programové i datové) po překladu na obou platformách a obou verzí implementace knihovny TUI jsou následovné:

Na platformě Atmel verze BIOS:

```
Program Memory Usage    :       7874 bytes   24,0 % Full
Data Memory Usage       :       1729 bytes   84,4 % Full
```

verze MENU:

```
Program Memory Usage    :       8236 bytes   25,1 % Full
Data Memory Usage       :       2020 bytes   98,6 % Full
```

Na platformě Microchip verze BIOS:

```
Program space        used  49B4h ( 18868) of  8000h bytes   ( 57.6%)
Data space           used   21Bh (   539) of   600h bytes   ( 35.1%)
```

verze MENU:

```
Program space        used  4B56h ( 19286) of  8000h bytes   ( 58.9%)
Data space           used   292h (   658) of   600h bytes   ( 42.8%)
```

Na uvedených výsledcích lze pozorovat rozdílný přístup překladačů dvou platforem k naprosto identickému kódu. AtmelStudio vkládá definici menu do paměti programové i do datové. Tento efekt by šlo zvrátit pouze zvláštními makry překladače, ovšem za cenu nekompatibility s ostatními překladači a jinými platformami. Odlišně se projevil překlad pro platformu Microchip, ve které se negativně odrazilo využití volně dostupného překladače XC8. U tohoto překladače je minimální optimalizace a proto výsledný kód v programové paměti zabírá více než dvojnásobek proti platformě Atmel.

Možnosti dalšího rozvoje

Jako každé softwarové dílo i knihovna TUI by se mohla dále rozvíjet. Některé nápady pro další funkcionalitu se vynořily už při její realizaci, ale vzhledem k rozsahu řešených funkcí se do vlastní realizace nedostaly. Jedním z nápadů byla změna hodnot podle přednastaveného výčtu (např. ano/ne, nízké/střední/vysoké). Další rozvoj by mohl zahrnovat přípravu rozhraní pro jiné způsoby komunikace knihovny, například po Ethernet síti nebo přes I^2C a jiné.

Použitá literatura

[1] LAWYER, David S. Text Terminal HOWTO, URL: <*http://www.linuxdoc.org/HOWTO/Text-Terminal-HOWTO.html*> [cit. 2015-01-22]

[2] PECHAL, Stanislav. Monolitické mikropočítače, *BEN - Technická literatura*, 1998, ISBN:80-86056-30-9

[3] OLIVKA, Petr. Komunikace s perifériemi, *Katedra informatiky FEI VŠB-TU Ostrava*, 2010, URL: <http://poli.cs.vsb.cz/edu/arp/down/komunikace.pdf>

[4] VT100 series Technical Manual, *Digital Equipment Corporation*, 1980, EK-VT100-TM-002, URL: <http://www.vt100.net/docs/vt100-tm/ek-vt100-tm-002.pdf>

[5] VT220 Programmer Reference Manual, *Digital Equipment Corporation*, 2001, EK-VT220-RM, URL: <http://www.vt100.net/docs/vt220-rm.zip>

[6] SCHMIDT, Robert. What Is The BIOS?, *Computing Basisc*, 1994

[7] MPLAB® XC8 C Compiler User's Guide, *Microchip Technology Inc.*, 2012-2015, DS50002053E, URL: <ww1.microchip.com/downloads/en/DeviceDoc/50002053E.pdf>

[8] PICkit™ 3 Debug Express PIC18F45K20 - MPLAB® C Lessons, *Michrochip Technology Inc.*, 2009, DS41370C, URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/41370C.pdf>

[9] ATMEL 8-bit AVR Microcontroller ATmega32 datasheet, *Atmel Corporation*, 2011, 2503Q-AVR-02/11, URL: <http://www.atmel.com/images/doc2503.pdf>

[10] PIC18F{23,24,25,26,43,44,45,46}K20 Data Sheet, *Michrochip Technology Inc.*, 2010, DS41303G, URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/41303G.pdf>

Seznam příloh

Příloha A: Dokumentace knihovny TUI
Příloha B: Tabulka ASCII
Příloha C: Seznam řídících kódů terminálu VT100 (List of Escape sequences) [4]
Příloha D: Morseova abeceda

Součástí bakalářské práce je CD.
Adresářová struktura přiloženého CD:

```
CDBP2015PG
    AtmelStudioSolution
        avrkit
            avrkit.c
            avrkit.h
        ucMorseCode
            ucBios.c
            ucBios.h
            ucCommon.c
            ucCommon.h
            ucLinkAVR.c
            ucLinkAVR.h
            ucMenu.c
            ucMenu.h
            ucMorseCode.cproj
            ucMorseCodeMain.c
        avr-kit.atsln
    MPLAB_X_IDE_Project
        ucMorseCode.X
            p18f45k20.h
            ucBios.c
            ucBios.h
            ucCommon.c
            ucCommon.h
            ucLinkPIC.c
            ucLinkPIC.h
            ucMenu.c
            ucMenu.h
            ucMorseCodeMain.c
        nbproject
            configurations.xml
            Makefile-default.mk
            Makefile-genesis.properties
            Makefile-impl.mk
            Makefile-local-default.mk
            Makefile-variables.mk
            Package-default.bash
            project.properties
            project.xml
    BP_GRE0056.pdf
```

Příloha A
DOKUMENTACE KNIHOVNY TUI

ucCommon

Výčty (enums):

Název	**ucResponseType**		
Popis	Určuje typ návratové hodnoty funkcí		
Hodnoty	*název*	*hodnota*	*popis*
	None	0	zobrazit text normálně
	Information	1	zobrazit podtržený text
	Warning	2	zobrazit text tučně
	Error	3	zobrazit text tučně a blikající
	BiosExit	4	ukončit menu

Název	**ucResponseType**		
Popis	Výčet pro rozpoznání kódů přijatých z terminálu		
Hodnoty	*název*	*hodnota*	*popis*
	Unknown	0	Neznámý
	Break	3	Přerušit
	BackSpace	8	Smazat před
	Enter	13	Potvrdit
	Escape	27	Odejít
	CursorUp	65	Nahoru
	CursorDown	66	Dolů
	CursorRight	67	Vpravo
	CursorLeft	68	Vlevo
	Delete	127	Smazat

Definice typů struktur:

Název	struct **ucResponse**		
Popis	Návratová hodnota funkcí		
Prvky	*typ*	*název*	*popis*
	enum ucResponseType	ResponseType	Typ zprávy
	char	*ResponseText	Návratová zpráva

Název	struct **ucMenu**		
Popis	Základní struktura menu obsahuje stránky menu (záložky). Menu musí mít alespoň jednu stránku.		
Prvky	*typ*	*název*	*popis*
	const char*	MenuTitle	Titulek menu
	const ucMenuPage*	MenuPages	Seznam stránek menu
	ucItemRead	Initialization	Funkce pro inicializaci menu
	ucItemRead	Dispose	Funkce pro destrukci menu
	unsigned char	PagesLine	Řádek, na kterém jsou zobrazeny tituly stránek (záložky)
	unsigned char	ItemsLine	Řádek, na kterém začíná první položka stránky menu
	unsigned char	DescrLine	Řádek, na kterém začíná popis položky menu
	unsigned char	EnterLine	Řádek, na kterém běží editor hodnot a zobrazení výsledků
	unsigned char	ValuesCol	Sloupec, od kterého jsou zobrazeny hodnoty položky menu
	char*	EnterPrompt	Text promptu zobrazeným před editovanou hodnotou
	ucReceiveCount	ReceiveCount	Funkce pro poctu znaku na vstupu rozhraní
	ucReceiveChar	ReceiveChar	Funkce pro načtení jednoho znaku ze vstupu
	ucTransmitChar	TransmitChar	Funkce pro odeslání jednoho znaku na výstup
	ucIntToDecStr	MenuIntToDecStr	Funkce pro překlad int na text
	ucRefreshCounterRead	RefreshCounterRead	Funkce pro načtení hodnoty čítače
	ucRefreshCounterReset	RefreshCounterReset	Funkce pro reset hodnoty čítače
	ucWaitOrSleep	MenuWaitOrSleep	Funkce pro čekání na určený čas

Název	struct **ucMenuPage**		
Popis	Stránka menu (záložka) obsahuje položky menu (nemusí mít ani jednu položku).		
Prvky	*typ*	*název*	*popis*
	char*	PageTitle	Název (titulek) stránky
	int	RefreshTime	Hodnota čítače, při které se má aktualizovat. 0=nikdy
	const ucMenuItem*	MenuItems	Seznam položek na stránce menu
	ucItemWrite	Write	Funkce stránky

Název	struct **ucMenuItem**
Popis	Položka menu na stránce. Druh je daný obsazením ukazatelů na funkce.

Druh	Read	Write	Popis
Text	---	---	Prázdný řádek nebo jen textový popisek
Pouze čtení	ano	---	Hodnota, kterou lze pouze číst
Čtení i zápis	ano	ano	Hodnota, kterou lze číst i měnit
Funkce	---	ano	Funkce, kterou lze z menu spustit

Prvky

typ	*název*	*popis*
char*	ItemName	Název položky na stránce menu
char*	ItemDescription	Popis položky na stránce menu
ucItemRead	Read	Funkce pro čtení textové reprezentace hodnoty
ucItemWrite	Write	Funkce pro zápis hodnoty nebo volání funkce

Definice typů ukazatelů na callback funkce:

Název	***ucItemRead**
Popis	Ukazatel na CallBack funkci zajišťující Inicializaci celého menu při startu nebo přečtení textové reprezentace aktuální hodnoty položky menu nebo destrukci menu
Argumenty	void
Návrat. hodnota	ucResponse - Obsahuje typ a zprávu (textovou reprezentaci hodnoty)

Název	***ucItemWrite**
Popis	Ukazatel na CallBack funkci zajišťující zápis nové hodnoty položky menu nebo volání funkce umístěné na položce menu
Argumenty	

typ	*název*	*popis*
char*	text	Obsahuje novou textovou reprezentaci hodnoty položky

Návrat. hodnota	ucResponse - Obsahuje typ a zprávu (zprávu o výsledku funkce)

Název	***ucReceiveCount**
Popis	Ukazatel na CallBack funkci zajišťující přečtení počtu znaků přijatých z terminálu a čekajících v zásobníku na další zpracování
Argumenty	void
Návrat. hodnota	int - Počet čekajících znaků v zásobníku

Název	***ucReceiveChar**
Popis	Ukazatel na CallBack funkci zajišťující přečtení jednoho znaku (nejdříve přijatého z terminálu) ze zásobníku
Argumenty	void
Návrat. hodnota	char - Přijatý znak

Název	***ucTransmitChar**		
Popis	Ukazatel na CallBack funkci pro odeslání znaku do připojeného terminálu		
Argumenty	*typ*	*název*	*popis*
	unsigned char	chr	Odesílaný znak
Návrat. hodnota	void		

Název	***ucRefreshCounterRead**
Popis	Ukazatel na CallBack funkci zajišťující přečtení hodnoty čítače určeného k automatické aktualizaci zobrazení hodnot položek na stránce menu
Argumenty	void
Návrat. hodnota	int - Aktuální hodnota čítače

Název	***ucRefreshCounterReset**
Popis	Ukazatel na CallBack funkci zajišťující vynulování čítače pro automatickou aktualizaci zobrazení
Argumenty	void
Návrat. hodnota	void

Název	***ucWaitOrSleep**		
Popis	Ukazatel na CallBack funkci zajišťující pozdržení vlákna procesu na přesně stanovenou dobu		
Argumenty	*typ*	*název*	*popis*
	int	miliseconds	Délka požadovaného zdržení v milisekundách
Návrat. hodnota	void		

Název	***ucIntToDecStr**		
Popis	Ukazatel na CallBack funkci zajišťující překlad čísla do textového řetězce v dekadické podobě		
Argumenty	*typ*	*název*	*popis*
	int	value	Číselná hodnota
	char*	text	Ukazatel na pole kam uložit výsledný tvar
Návrat. hodnota	void		

Veřejné funkce:

Název	**SendText**		
Popis	Funkce pro odeslání znaků z pole ukončeného znakem 0x00		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na konstantní strukturu celého menu
	const char*	buffer	Ukazatel na pole znaků pro odeslání
Návrat. hodnota	void		

Veřejné funkce (řídící terminál):

Název	**ClearDisplay**		
Popis	Funkce provede kompletní smazání displeje terminálu		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na konstantní strukturu celého menu
Návrat. hodnota	void		

Název	**SetPosition**		
Popis	Funkce provede nastavení kurzoru na zadanou pozici displeje		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na konstantní strukturu celého menu
	unsigned char	yLine	Požadovaný řádek
	unsigned char	xChar	Požadovaný sloupec
Návrat. hodnota	void		

Název	**ClearLineFromCursor**		
Popis	Funkce zajistí smazání řádku displeje od pozice kurzoru do konce řádku		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na konstantní strukturu celého menu
Návrat. hodnota	void		

Název	**ClearLine**		
Popis	Funkce provede smazání celého požadovaného řádku		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na konstantní strukturu celého menu
	unsigned char	yLine	Požadovaný řádek
Návrat. hodnota	void		

Název	**SetAttribute**		
Popis	Funkce určí atributy pro následně odeslaný text		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
	char	bold	Zobrazit text tučně
	char	underscored	Zobrazit podtržený text
	char	blinking	Zobrazit blikající text
	char	invert	Zobrazit text inverzně
Návrat. hodnota	void		

Název	**SetAttributeByType**		
Popis	Funkce určí atributy pro následný text podle typu návratové hodnoty		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
	ucResponseType	responseType	Zobrazit text podle typu
Návrat. hodnota	void		

ucBios

Veřejné funkce:

Název	**RunBios**		
Popis	Funkce provádějící kompletní servis menu		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
	unsigned char	page	Aktivní stránka při startu menu (0based)
	signed char	item	Aktivní položka při startu menu (0based)
Návrat. hodnota	char* - Vrací zprávu z návratové hodnoty poslední funkce menu		

Privátní funkce:

Název	**_WaitBiosCommand**		
Popis	Funkce z cyklem očekávající příkaz z terminálu a případně automatickým obnovením hodnot položek menu zobrazených na aktuální stránce		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
	unsigned char	actualPage	Index aktuální stránky (0based)
Návrat. hodnota	enum ucCommandType – Rozeznaný poslední příkaz z terminálu		

Název	**_EditBiosItemValue**		
Popis	Funkce zajistí editaci zadávané hodnoty položky menu		
Argumenty	typ	název	popis
	const ucMenu*	menu	Odkaz na strukturu celého menu
	char*	oldValue	Ukazatel na pole znaků s původní hodnotou položky
	char*	newValue	Ukazatel na pole znaků pro novou hodnotu položky
Návrat. hodnota	char* - Délku řetězce nově zadané hodnoty, nebo -1 pokud bylo zadávání přerušeno klávesou ESC		

Název	**_WriteBiosItemValue**		
Popis	Funkce spustí zápisovou funkci položky menu a případně zobrazí výsledek		
Argumenty	typ	název	popis
	const ucMenu*	menu	Odkaz na strukturu celého menu
	unsigned char	page	Aktuální stránka menu (0based)
	signed char	item	Aktuální položka menu (0based)
	char*	newValue	Ukazatel na pole znaků s novou hodnotou položky
Návrat. hodnota	ucResponse* - Vrací návratovou hodnotu zápisové funkce položky		

Název	**_RefreshBiosPage**		
Popis	Funkce zobrazí / obnoví stránku menu		
Argumenty	typ	název	popis
	const ucMenu*	menu	Odkaz na strukturu celého menu
	unsigned char	page	Aktuální stránka menu (0based)
	signed char	Actualitem	Aktuální položka menu (0based) -1 - aktuální je samotná záložka stránky >= 0 - aktuální položka menu na stránce
Návrat. hodnota	void		

Název	**_RefreshBiosPageValues**		
Popis	Funkce zobrazí / obnoví pouze hodnoty položek na stránce menu		
Argumenty	typ	název	popis
	const ucMenu*	menu	Odkaz na strukturu celého menu
	unsigned char	page	Aktuální stránka menu (0based)
Návrat. hodnota	void		

Název	**_RefreshBiosPageItemLine**		
Popis	Funkce pro zobrazení celého řádku položky menu i s případnou hodnotou		
Argumenty	typ	název	Popis
	const ucMenu*	menu	Odkaz na strukturu celého menu
	unsigned char	page	Aktuální stránka menu (0based)
	signed char	item	Zobrazovaná položka menu (0based)
	char	actual	Příznak, zda se jedná o aktuální položku
Návrat. hodnota	ucResponse* - Vrací návratovou hodnotu zápisové funkce položky		

Název	**_RefreshBiosPageItemValue**		
Popis	Funkce pro zobrazení hodnoty položky menu		
Argumenty	typ	název	Popis
	const ucMenu*	menu	Odkaz na strukturu celého menu
	unsigned char	page	Aktuální stránka menu (0based)
	signed char	item	Zobrazovaná položka menu (0based)
Návrat. hodnota	void		

ucMenu

Veřejné funkce:

Název	**DoMenuActions**		
Popis	Funkce provádějící obsluhu jedné akce menu		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
Návrat. hodnota	void		

Privátní funkce:

Název	**_WaitingForMenuCommand**		
Popis	Funkce vyhodnocující příkaz z terminálu a případně automatickým obnovením hodnot položek menu zobrazených na aktuální stránce		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
Návrat. hodnota	void		

Název	**_RefreshMenuPageItem**		
Popis	Funkce zobrazení / obnovení stránky menu - postupně po prvcích		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
Návrat. hodnota	void		

Název	**_RefreshMenuPageValues**		
Popis	Funkce postupně zobrazí / obnoví pouze hodnoty stránky menu		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
Návrat. hodnota	void		

Název	**_RefreshMenuPageItemValue**		
Popis	Funkce pro zobrazení hodnoty jediné obnovované položky na stránce		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
Návrat. hodnota	void		

Název	**_WriteMenuItemValue**		
Popis	Funkce zavolá zápisovou funkci položky a zobrazí případnou odpověď		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
	char*	newValue	Ukazatel na pole znaků s nov. hodnotou
Návrat. hodnota	void		

Název	**_RefreshMenuPageItemValue**		
Popis	Funkce postupně zajistí editaci řetězce zadávané hodnoty položky		
Argumenty	*typ*	*název*	*popis*
	const ucMenu*	menu	Odkaz na strukturu celého menu
Návrat. hodnota	void		

PŘÍLOHA B
TABULKA ASCII

Dec	Hex	Zkratka	Význam
0	00	NULL	NULL character
1	01	SOH	Start of Header
2	02	STX	Start of Text
3	03	ETX	End of Text
4	04	EOT	End of Transmiss
5	05	ENQ	Enquiry
6	06	ACK	Acknowledge
7	07	BEL	Bell
8	08	BS	Backspace
9	09	HT	Horizontal Tab
10	0A	LF	Line Feed
11	0B	VT	Vertical Tab
12	0C	FF	Form Feed
13	0D	CR	Carriage Return
14	0E	SO	Shift Out
15	0F	SI	Shift In
16	10	DLE	Data Link Escape
17	11	DC1	(XOn)
18	12	DC2	
19	13	DC3	(XOff)
20	14	DC4	
21	15	NAK	Neg.Acknowledge
22	16	SYN	Synchronous Idle
23	17	ETB	End of TransBlock
24	18	CAN	Cancel
25	19	EM	End of Medium
26	1A	SUB	Substitute
27	1B	ESC	Escape
28	1C	FS	FileSeparator
29	1D	GS	Group Separator
30	1E	RS	Record Separator
31	1F	US	Unit Separator

Dec	Hex	Znak
32	20	space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Znak
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Znak
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	Del

Příloha C
SEZNAM ŘÍDÍCÍCH KÓDŮ TERMINÁLU VT100 (LIST OF ESCAPE SEQUENCES)

Table A-11 VT100 Escape Sequences Summary

Feature	VT52 Compatible Mode
Cursor Up	ESC A
Cursor Down	ESC B
Cursor Right	ESC C
Cursor Left	ESC D
Select Special Graphics character set	ESC F
Select ASCII character set	ESC G
Cursor to home	ESC H
Reverse line feed	ESC I
Erase to end of screen	ESC J
Erase to end of line	ESC K
Direct cursor address	ESC Y\lc (see note 1)
Identify	ESC Z (see note 2)
Enter alternate keypad mode	ESC =
Exit alternate keypad mode	ESC >
Graphics processor on	ESC 1 (see note 3)
Graphics processor off	ESC 2 (see note 3)
Enter ANSI mode	ESC <

Note 1
Line and column numbers for direct cursor address are single character codes whose values are the desired number plus 37_8. Line and column numbers start at 1.

Note 2
Response to ESC Z is ESC / Z.

Note 3
Ignored if no graphics processor option is installed in the VT100.

Erasing

From cursor to end of line	ESC [K or ESC [0 K
From beginning of line to cursor	ESC [1 K
Entire line containing cursor	ESC [2 K
From cursor to end of screen	ESC [J or ESC [0 J
From beginning of screen to cursor	ESC [1 J
Entire screen	ESC [2 J

Programmable LEDs
ESC [Ps; Ps;...Ps q

Ps are selective parameters separated by semicolons (073_8) and executed in order, as follows.

0 or none	All LEDs off
1	LED #1 on
2	LED #2 on
3	LED #3 on
4	LED #4 on

Any other parameter values are ignored.

Character Sets (G0 and G1 Designators)
The G0 and G1 character sets are designated as follows:

Character Set	G0	G1
United Kingdom (UK)	ESC (A	ESC) A
United States (USASCII)	ESC (B	ESC) B
Special graphics characters and line drawing set	ESC (0	ESC) 0
Alternate character ROM	ESC (1	ESC) 1
Alternate character ROM special graphics characters	ESC (2	ESC) 2

Scrolling Region
ESC [Pt ; Pb r

Pt is the number of the top line of the scrolling region. Pb is the number of the bottom line of the scrolling region and must be greater than Pc.

Tab Stops

Set tab at current column	ESC H
Clear tab at current column	ESC [g or ESC [0 g
Clear all tabs	ESC [3 g

Modes

	To Set		To Reset	
Mode Name	Mode	Sequence	Mode	Sequence
Line feed/new line	New line	ESC [20h	Line feed	ESC [20l
Cursor key mode	Application	ESC [?1h	Cursor	ESC [?1l
ANSI/VT52	ANSI	N/A	VT52	ESC [?2l
Column mode	132 column	ESC [?3h	80 column	ESC [?3l
Scrolling mode	Smooth	ESC [?4h	Jump	ESC [?4l
Screen mode	Reverse	ESC [?5h	Normal	ESC [?5l
Origin mode	Relative	ESC [?6h	Absolute	ESC [?6l
Wraparound	On	ESC [?7h	Off	ESC [?7l
Auto repeat	On	ESC [?8h	Off	ESC [?8l
Interlace	On	ESC [?9h	Off	ESC [?9l
Graphic proc. opt.	On	ESC]	Off	ESC 2
Keypad mode	Application	ESC =	Numeric	ESC >

Reports

Cursor Position Report

Invoked by	ESC [6 n
Response is	ESC [Pl ; Pc R (Pl = line number and Pc = column number)

Status Report

Invoked by	ESC [5 n
Response is	ESC [0 n (terminal ok) ESC [3 n (terminal not ok)

What Are You

Invoked by	ESC [c or ESC [O c
Response is	ESC [? 1 ; Ps c

Ps is the "option present" parameter with the following meaning.

Ps	Meaning
0	Base VT100, no options
1	Processor option (STP)
2	Advanced video option (AVO)
3	AVO and STP
4	Graphics processor option (GO)
5	GO and STP
6	GO and AVO
7	GO, STP, and AVO

Alternately invoked by ESC Z (not recommended). Response is the same.

Reset

Reset causes the power-up reset routine to be excecuted.

ESC c

Confidence Tests

Fill screen with "Es"	ESC # 8
Invoke test(s)	ESC [2 ; Ps y

Ps is the parameter indicating the test to be done and is a decimal number computed by taking the "weight" indicated for each desired test and adding them together.

Test	Weight
Power-up self test (ROM checksum, RAM, NVR, keyboard and AVO if installed)	1
Data loopback	2 (Loopback connector required)
Repeat selected tests(s) indefinitely (until failure or power off)	8

Valid ANSI Mode Escape Sequences (Detailed)

Definitions – The following listing defines the basic elements of the ANSI mode escape sequences.

Control sequence introducer (CSI) – An escape sequence that provides supplementary controls and is itself a prefix affecting the interpretation of a limited number of contiguous characters. In the VT100 the CSI is ESC [.

Parameter – (1) A string of one or more decimal characters that represent a single value; (2) The value so represented.

Numeric Parameter – A string of characters that represent a number, designated by Pn. Pn has a range of 0 (60_8) to 9 (71_8).

Selective Parameter – A string of characters that selects a subfunction from a specified list of subfunctions, designated by Ps. In general, a control sequence with more than one selective parameter causes the same effect as several control sequences, each with one selective parameter, e.g., CSI Psa; Psb; Psc F is identical to CSI Psa F CSI Psb F CSI Psc F.

Ps is a string of zero or more characters with a range of 0 (60_8) to 9 (71_8) with each selective parameter separated from the other by; (73_8).

Default – A function-dependent value assumed when no explicit value, or a value of 0, is specified.

Final character – A character whose bit combination terminates an escape or control sequence.

Escape sequences – All of the following escape and control sequences are transmitted from the host computer to the VT100 unless otherwise noted. All of the escape sequences are a subset **of those specified in ANSI** X3.64 1977 and ANSI X3.41 1974.

CPR Cursor Position Report – VT100 to Host

Format: ESC [Pn; Pn R **default values: 1**

The CPR sequence reports the active position by means of the parameters. This sequence has two parameter values, the first specifying the line and the second specifying the column. The default condition with no parameters present, or parameters of 0, is equivalent to a cursor at home position.

Numbering of lines depends on the state of the Origin mode (DECOM).

This control sequence is solicited by a device status report (DSR) sent from the host.

CUB Cursor Backward – Host to VT100 and VT100 to Host

Editor Function; format: ESC [Pn D **default value: 1**

The CUB sequence moves the active position to the left. The distance moved is determined by the parameter. If the parameter value is zero or one, the active position moves one position to the left. If the parameter value is n, the active position moves n positions to the left. If an attempt is made to move the cursor to the left of the left margin, the cursor stops at the left margin.

CUD Cursor Down – Host to VT100 and VT100 to Host

Editor Function; format: ESC [Pn B **default value: 1**

The CUD sequence moves the active position downward without altering the column position. The number of lines moved is determined by the parameter. If the parameter value is zero or one, the active position moves one line downward. If the parameter value is n, the active position moves n lines downward. If an attempt is made to move the cursor below the bottom margin, the cursor stops at the bottom margin.

CUF Cursor Forward – Host to VT100 and VT100 to Host

Editor Function; format: ESC [Pn C **default value: 1**

The CUF sequence moves the active position to the right. The distance moved is determined by the parameter. A parameter value of zero or one moves the active position one position to the right. A parameter value of n moves the active position n positions to the right. If an attempt is made to move the cursor to the right of the right margin, the cursor stops at the right margin.

CUP Cursor Position

Editor Function; format: ESC [Pn; Pn H **default value: 1**

The CUP sequence moves the active position to the position specified by the parameters. This sequence has two parameter values, the first specifying the line position and the second specifying the column position. A parameter value of zero or one for the first or second parameter moves the active position to the first line or column in the display, respectively. The default condition with no parameters present is equivalent to a cursor to home action. In the VT100, this control behaves identically with its format effector counterpart, HVP.

The numbering of lines depends on the state of the Origin mode (DECOM).

CUU Cursor Up – Host to VT100 and VT100 to Host

Editor Function; format: ESC [Pn A **default value: 1**

This sequence moves the active position upward without altering the column position. The number of lines moved is determined by the parameter. A parameter value of zero or one moves the active position one line upward. A parameter value of n moves the active position n lines upward. If an attempt is made to move the cursor above the top margin, the cursor stops at the top margin.

DA Device Attributes

Format: ESC [Pn c **default value: 0**

1. The host requests the VT100 to send a device attributes (DA) control sequence to identify itself by sending the DA control sequence with either no parameter or a parameter of 0.

2. Response to the request described above (VT100 to host) is generated by the VT100 as a DA control sequence with the numeric parameters as follows:

Option Present	Sequence Sent
No options	ESC [?1;0c
Processor option (STP)	ESC [?1;1c
Advanced video option (AVO)	ESC [?1;2c
AVO and STP	ESC [?1;3c
Graphics option (GO)	ESC [?1;4c
GO and STP	ESC [?1;5c
GO and AVO	ESC [?1;6c
GO, STP, and AVO	ESC [?1;7c

DECALN Screen Alignment Display (DEC Private)

Format: ESC #8

This command fills the entire screen area with uppercase Es for screen focus and alignment. This command is used by DEC manufacturing and Field Service personnel.

DECANM ANSI/VT52 Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes only VT52 compatible escape sequences to be interpreted and executed. The set state causes only ANSI "compatible" escape and control sequences to be interpreted and executed.

DECARM Auto Repeat Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes no keyboard keys to autorepeat. The set state causes certain keyboard keys to autorepeat.

DECAWM Autowrap Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes any displayable characters received when the cursor is at the right margin to replace any previous characters there. The set state causes these characters to advance to the start of the next line, doing a scroll up if required and permitted.

DECCOLM Column Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes a maximum of 80 columns on the screen. The set state causes a maximum of 132 columns on the screen.

DECCKM Cursor Keys Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. This mode is only effective when the terminal is in keypad application mode (see DECKPAM) and the ANSI/VT52 mode (DECANM) is set (see DECANM). Under these conditions, if the cursor key mode is reset, the four cursor function keys send ANSI cursor control commands. If cursor key mode is set, the four cursor function keys send application functions.

DECDHL Double-Height Line (DEC Private)

Format: Top Half: ESC #3, Bottom Half: ESC #4

These sequences cause the line containing the active position to become the top or bottom half of a double-height double-width line. The sequences must be used in pairs on adjacent lines and the same character output must be sent to both lines to form full double-height characters. If the line was single-width single-height, all characters to the right of the center of the screen are lost. The cursor remains over the same character position unless it would be to the right of the right margin, in which case it is moved to the right margin.

DECDWL Double-Width Line (DEC Private)

Format: ESC #6

This sequence causes the line that contains the active position to become double-width, single-height. If the line was single-width, single-height, all characters to the right of screen center are lost. The cursor remains over the same character position unless it would be to the right of the right margin, in which case, it is moved to the right margin.

DECGOFF Graphics Processor OFF (DEC Private)

Format: ESC 2

Turn off the graphics processor.

NOTE
Some DEC hardcopy terminals interpret this private escape sequence as CLEAR TABS.

DECGON Graphics Processor On (DEC Private)

Format: ESC 1

All subsequent characters are interpreted as commands or data to the graphics processor option. The terminal remains in this mode until the graphics processor off command is received. This command is ignored if no graphics processor option is installed.

NOTE
Some DEC hardcopy terminals interpret this private escape sequence as SET TAB.

DECHCP Hard Copy (DEC Private)

Format: ESC #7

This sequence causes the screen to cease updating and freeze while the hardcopy output is enabled. The screen resumes normal operation when the hardcopy has been completed. This command is ignored if no hardcopy option is installed.

DECID Identify Terminal (DEC Private)

Format: ESC Z

This sequence causes the same response as the ANSI device attributes (DA). DECID will not be supported in future DEC terminals, therefore, DA should be used by any new software.

DECINLM Interlace Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state (noninterlace) causes the video processor to display 240 scan lines per frame. The set state (interlace) causes the video processor to display 480 scan lines per frame. There is no increase in character resolution.

DECKPAM Keypad Application Mode (DEC Private)

Format: ESC =

Auxiliary keypad keys and cursor control keys transmit escape sequences.

DECKPNM Keypad Numeric Mode (DEC Private)

Format: ESC >

Auxiliary keypad keys send ASCII codes corresponding to the characters on the keys. Cursor control keys send cursor controls.

DECLL Load LEDs (DEC Private)

Format: ESC [Ps q **default value: 0**

Load the four programmable LEDs on the keyboard according to the parameter(s).

Parameter	Meaning
0	Clear LEDs 1 through 4
1	Light LED 1
2	Light LED 2
3	Light LED 3
4	Light LED 4

LED numbers are indicated on the keyboard.

DECOM Origin Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes the origin to be at the upper-left character position on the screen. Line numbers are therefore independent of current margin settings. The cursor may be positioned outside the margins with a cursor position (CUP) or horizontal and vertical position (HVP) control.

The set state causes the origin to be at the upper-left character position within the margins. Line numbers are therefore relative to the current margin settings. The cursor cannot be positioned outside the margins.

The cursor moves to the new home position when this mode is set or reset.

Lines and columns are numbered consecutively, with the origin being line 1, column 1.

DECRC Restore Cursor (DEC Private)

Format: ESC 8

This sequence causes the previously saved cursor position, graphic rendition, and character set to be restored.

DECREPTPARM Report Terminal Parameters

Format: ESC [<sol>; <par>; <nbits>; <xspeed>; <rspeed>; <clkmul>; <flags>x

These sequence parameters are explained below in the DECREQTPARM sequence.

DECREQTPARM Request Terminal Parameters

Format: ESC [<sol>x

This sequence is sent by the terminal controller to notify the host of the status of selected terminal parameters. The status sequence may be sent when requested by the host or at the terminal's discretion. DECREPTPARM is sent upon receipt of a DECREQTPARM.

The meanings of the sequence parameters are:

Parameter	Value	Meaning
<sol>	0 or none	This message is a request (DECREQTPARM). The terminal is allowed to send unsolicited reports.
	1	This message is a request. From now on the terminal may only report in response to a request.
	2	This message is a report (DECREPTPARM).
	3	This message is a report and the terminal is only reporting on request.
<par>	1	No parity set
	4	Parity is set and odd
	5	Parity is set and even
<nbits>	1	8 bits per character
	2	7 bits per character
<xspeed>		Bits per second
<rspeed>	0	50
	8	75
	16	110
	24	134.5
	32	150
	40	200
	48	300
	56	600
	64	1200
	72	1800
	80	2000
	88	2400
	96	3600
	104	4800
	112	9600
	120	19,200
<clkmul>	1	The bit rate multiplier is 16.
<flags>	0–15	This value communicates the four switch values in block 5 of SET UP B, which are only visible to the user when an STP option is installed. These bits may be assigned for an STP device. The four bits are a decimal-encoded binary number.

DECSC Save Cursor (DEC Private)

Format: ESC 7

This sequence causes the cursor position, graphic rendition, and character set to be saved. (See DECRC.)

DECSCLM Scrolling Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes scrolls to "jump" instantaneously. The set state causes scrolls to be "smooth" at a maximum rate of six lines per second.

DECSCNM Screen Mode (DEC Private)

This is a private parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes the screen to be black with white characters. The set state causes the screen to be white with black characters.

DECSTBM Set Top and Bottom Margins (DEC Private)

Format: ESC [Pn; Pn r **default values: (see below)**

This sequence sets the top and bottom margins to define the scrolling region. The first parameter is the line number of the first line in the scrolling region. The second parameter is the line number of the bottom line in the scrolling region. Default is the entire screen (no margins). The minimum size of the scrolling region allowed is two lines, i.e., the top margin must be less than the bottom margin.

DECSWL Single-Width Line (DEC Private)

Format: ESC #5

This causes the line that contains the active position to become single-width, single-height. The cursor remains on the same character position. This is the default condition for all new lines on the screen.

DECTST Invoke Confidence Test

Format: ESC [2 ; Ps y

Ps is the parameter indicating the test to be done. Ps is computed by taking the weight indicated for each desired test and adding them together. If Ps is 0 no test is performed but the VT100 is reset.

Test	Weight
Power up self-test (ROM checksum, RAM, NVR keyboard and AVO if installed)	1
Data Loopback	2 (loopback connector required)
Repeat selected test(s) indefinitely (until failure or power off)	8

DSR Device Status Report

Format: ESC [Ps r **default value: 0**

Requests and reports the general status of the VT100 according to the following parameter(s).

Parameter	Meaning
0	Response from VT100 – ready, no malfunctions detected (default)
3	Response from VT100 – malfunction – retry
5	Command from host – please report status (using a DSR control sequence)
6	Command from host – please report active position (using a CPR control sequence)

DSR with a parameter value of 0 or 3 is always sent as a response to a requesting DSR with a parameter value of 5.

ED Erase In Display

Editor Function; format: ESC [Ps J **default value: 0**

This sequence erases some or all characters in the display according to the parameter. Any complete line erased by this sequence returns that line to single-width mode.

Parameter	Meaning
0	Erase from the active position to the end of the screen, inclusive (default).
1	Erase from start of the screen to the active position, inclusive.
2	Erase all of the display — all lines are erased, changed to single-width, and the cursor does not move.

EL Erase In Line

Editor Function; format: ESC [Ps K **default value: 0**

Erases some or all characters in the active line according to the parameter.

Parameter	Meaning
0	Erase from the active position to the end of the line, inclusive (default).
1	Erase from the start of the screen to the active position, inclusive.
2	Erase all of the line, inclusive.

HTS Horizontal Tabulation Set

Format Effector; format: ESC H

Set one horizontal stop at the active position.

HVP Horizontal and Vertical Position

Format Effector, format: ESC [Pn; Pn f **default value: 1**

Moves the active position to the position specified by the parameters. This sequence has two parameter values; the first specifies the line position, the second specifies the column. A parameter value of either zero or one causes the active position to move to the first line or column in the display, respectively. The default condition with no parameters present moves the active position to the home position. In the VT100, this control behaves identically with its editor function counterpart, CUP. Numbering lines and columns depends on the reset or set state of the origin mode (DECOM).

IND Index

Format Effector; format: ESC D

This sequence causes the active position to move downward one line without changing column position. If the active position is at the bottom margin, a scroll up is performed.

LNM Line Feed/New Line Mode

This is a parameter applicable to set mode (SM) and reset mode (RM) control sequences. The reset state causes the interpretation of the line feed (LF), defined in ANSI Standard X3.4-1977, to imply only vertical movement of the active position and causes the return key (CR) to send the single code CR. The set state causes the LF to imply movement to the first position of the following line and causes the return key to send the two codes (CR, LF). This is the new line (NL) option.

This mode does not affect the index (IND), or next line (NEL) format effectors.

NEL Next Line

Format Effector; format: ESC E

This sequence causes the active position to move to the first position on the next line downward. If the active position is at the bottom margin, a scroll up is performed.

RI Reverse Index

Format Effector; format: ESC M

Moves the active position to the same horizontal position on the preceding line. If the active position is at the top margin, a scroll down is performed.

RIS Reset to Initial State

Format: ESC c

Resets the VT100 to its initial state, i.e., the state it has after it is powered on. This also causes the execution of the power-up self test and signal INIT H to be asserted briefly.

RM Reset Mode

Format: ESC [Ps ;Ps ; ...; Ps l **default value: none**

Resets one or more VT100 modes as specified by each selective parameter in the parameter string. Each mode to be reset is specified by a separate parameter. (See modes and set mode (SM) following this section.)

SCS Select Character Set

The appropriate G0 and G1 character sets are selected from the five possible character sets. G0 and G1 are selected by codes SI and SO (shift in and shift out) respectively.

G0 Sets Sequence	G1 Sets Sequence	Meaning
ESC (A	ESC) A	United Kingdom set
ESC (B	ESC) B	ASCII set
ESC (0	ESC) 0	Special graphics
ESC (1	ESC) 1	Alternate character ROM Standard character set
ESC (2	ESC) 2	Alternate character ROM Special graphics

The United Kingdom and ASCII sets conform to the "ISO international register of character sets to be used with escape sequences." The other sets are private character sets. Special graphics means that the graphic characters for the codes 137_8 to 176_8 are replaced with other characters. The specified character set will be used until another SCS is received.

SGR Select Graphic Rendition

Format Effector; format: ESC [Ps; ...; Ps m **default value: 0**

Invoke the graphic rendition specified by the parameter(s). All following characters transmitted to the VT100 are rendered according to the parameter(s) until the next occurrence of SGR.

Parameter	Meaning
0	Attributes off
1	Bold or increased intensity
4	Underscore
5	Blink
7	Negative (reverse) image

All other parameter values are ignored.

Without the advanced video option only one character attribute is possible as determined by the cursor selection. In that case specifying either the underscore or the reverse attribute activates the currently selected attribute.

SM Set Mode

Format: ESC [Ps; ...; Ps h **default value: none**

Causes one or more modes to be set within the VT100 as specified by each selective parameter in the parameter string. Each mode to be set is specified by a separate parameter. A mode is considered set until it is reset by a reset mode (RM) control sequence.

TBC Tabulation Clear

Format Effector; format: ESC [Ps g **default value: 0**

Parameter	Meaning
0	Clear the horizontal tab stop at the active position (the default case).
3	Clear all horizontal tab stops.

Any other parameter values are ignored.

MODES The following VT100 modes may be changed with set mode (SM) and reset mode (RM) controls.

ANSI Specified Modes

Parameter	Mode Mnemonic	Function
0		Error (ignored)
20	LNM	Line feed new line mode

DEC Private Modes

If the first character in the parameter string is ? (77_8), the parameters are interpreted as DEC private parameters according to the following modes:

Parameter	Mode Mnemonic	Function
0		Error (ignored)
1	DECCKM	Cursor key
2	DECANM	ANSI/VT52
3	DECCOLM	Column
4	DECSCLM	Scrolling
5	DECSCNM	Screen
6	DECOM	Origin
7	DECAWM	Auto wrap
8	DECARM	Auto repeating
9	DECINLM	Interlace

Any other parameter values are ignored.

The following modes, specified in ANSI X3.64-1977 standard, may be considered to be permanently set, permanently reset, or not applicable, as noted. Refer to that standard for further information concerning these modes.

Mnemonic	Function	State
CRM	Control representation	Reset
EBM	Editing boundary	Reset
ERM	Erasure	Set
FEAM	Format effector action	Reset
FETM	Format effector transfer	Reset
GATM	Guarded area transfer	NA
HEM	Horizontal editing	NA
IRM	Insertion-replacement	Reset
KAM	Keyboard action	Reset
MATM	Multiple area transfer	NA
PUM	Positioning unit	Reset
SATM	Selected area transfer	NA
SRTM	Status reporting transfer	Reset
TSM	Tabulation stop	Reset
TTM	Transfer termination	NA
VEM	Vertical editing	NA

Valid VT52 Mode Escape Sequences (detailed)

Graphics Processor On

Format: ESC 1

All subsequent characters are interpreted as commands to the graphics option until the graphics processor off command is received. This sequence is ignored if no graphics processor is installed.

Graphics Processor Off

Format: ESC 2

Turn off the graphics processor.

Cursor Up

Format: ESC A

Move the active position upward one position without altering the horizontal position. If an attempt is made to move the cursor above the top margin, the cursor stops at the top margin.

Cursor Down

Format: ESC B

Move the active position downward one position without altering the horizontal position. If an attempt is made to move the cursor below the bottom margin, the cursor stops at the bottom margin.

Cursor Right

Format: ESC C

Move the active position to the right. If an attempt is made to move the cursor to the right of the right margin, the cursor stops at the right margin.

Cursor Left

Format: ESC D

Move the active position one position to the left. If an attempt is made to move the cursor to the left of the left margin, the cursor stops at the left margin.

Enter Graphics Mode

Format: ESC F

Causes the special graphics character set to be used.

NOTE
The special graphics characters in the VT100 are different from those in the VT52.

Exit Graphics Mode

Format: ESC G

This sequence causes the standard ASCII character set to be used.

Cursor to Home

Format: ESC H

Move the cursor to the home position.

Reverse Line Feed

Format: ESC I

Move the active position upward one position without altering the column position. If the active position is at the top margin, a scroll down is performed.

Erase to End of Screen

Format: ESC J

Erase all characters from the active position to the end of the screen. The active position is not changed.

Erase to End of Line

Format: ESC K

Erase all characters from the active position to the end of the current line. The active position is not changed.

Direct Cursor Address

Format: ESC Y line column

Move the cursor to the specified line and column. The line and column numbers are sent as ASCII codes whose values are the number plus 037_8. For example, 040_8 refers to the first line or column, 050_8 refers to the eighth line or column, etc.

Identify

Format: ESC Z

Causes the terminal to send its identifier escape sequence to the host. The sequence is: ESC / Z.

NOTE
Information regarding options must be obtained in ANSI mode, using the device attributes (DA) control sequence.

Enter Alternate Keypad Mode

Format: ESC =

Auxiliary keypad keys send unique identifiable escape sequences for use by applications programs.

Exit Alternate Keypad Mode

Format: ESC >

Auxiliary keypad keys send ASCII codes for functions or characters on the key.

Enter ANSI Mode

Format: ESC <

All subsequent escape sequences will be interpreted according to ANSI Standards X3.64-1977 and X3.41-1974. The VT52 escape sequence in this section will not be recognized.

Příloha D:
MORSEOVA ABECEDA

Mezinárodní znaky, které využívá prototypová aplikace (+ česká mnemotechnická pomůcka):

Znak	Kód	Pomůcka	Znak	Kód	Pomůcka
A	·—	akát	**N**	—·	národ
B	—···	blýskavice	**O**	———	ó náš háj
C	—·—·	cílovníci	**P**	·——·	papírníci
D	—··	dálava	**Q**	——·—	kvílí orkán
E	·	erb	**R**	·—·	rarášek
F	··—·	filiálka	**S**	···	sekera
G	——·	Grónská zem	**T**	—	tón
H	····	holubice	**U**	··—	uličník
I	··	ibis	**V**	···—	vyvolený
J	·———	junácká hůl	**W**	·——	waltrův vůz
K	—·—	království	**X**	—··—	Xénokratés
L	·—··	lední hokej	**Y**	—·——	ý se ztrácí
M	——	mává	**Z**	——··	zrádná žena

Znak	Kód	Znak	Kód
0	—————	**5**	·····
1	·————	**6**	—····
2	··———	**7**	——···
3	···——	**8**	———··
4	····—	**9**	————·